گزارش تحلیلی جامع برای تولید سیستم بودجه‌ریزی

با توجه به مدل‌های ارائه‌شده (BudgetPeriod, BudgetAllocation, ProjectBudgetAllocation, BudgetTransaction, PaymentOrder, Payee, TransactionType, BudgetReallocation, BudgetSettings, BudgetHistory) و درخواست شما برای تکمیل و بهبود سیستم بودجه‌ریزی، در این گزارش تحلیلی همه جنبه‌ها را با دقت بررسی کرده‌ام، نقاط قوت و ضعف را تحلیل می‌کنم، پیشنهاداتم را برای بهبود ارائه می‌دهم و موارد درخواستی مثل فرم تنظیمات، انتقال محاسبات به فایل جداگانه، پیش‌بینی بودجه با هوش مصنوعی، برنامه‌ریزی سناریو، مسیر حسابرسی و مدیریت هزینه‌های متعارف را اضافه می‌کنم. این گزارش چندین بار بازبینی شده تا از جامعیت و دقت آن مطمئن شوم.

۱. بررسی مدل‌های موجود

مدل‌های فعلی یک ساختار قوی برای مدیریت بودجه از سطح کلان (دفتر مرکزی) تا جزئیات پروژه‌ها، پرداخت‌ها و تراکنش‌ها فراهم می‌کنند. در ادامه نقاط قوت و ضعف آن‌ها را تحلیل می‌کنم:

نقاط قوت

انعطاف‌پذیری در تخصیص بودجه: مدل‌های BudgetAllocation و ProjectBudgetAllocation امکان تخصیص بودجه به سازمان‌ها، پروژه‌ها و زیرپروژه‌ها را با جزئیات فراهم می‌کنند.

کنترل قفل و هشدار: فیلدهای locked\_percentage, warning\_threshold و warning\_action در BudgetPeriod و BudgetAllocation ابزارهای قدرتمندی برای مدیریت محدودیت‌ها و اعلانات هستند.

ردیابی تغییرات: BudgetTransaction و BudgetHistory تاریخچه تغییرات بودجه را به‌خوبی ثبت می‌کنند.

مدیریت پرداخت‌ها: PaymentOrder و Payee فرآیند پرداخت را کامل و شفاف می‌کنند.

تنظیمات پویا: BudgetSettings امکان اعمال تنظیمات خاص در سطوح مختلف (دوره، تخصیص، پروژه) را می‌دهد.

نقاط ضعف و نکات قابل بهبود

ناهماهنگی در remaining\_amount:

در BudgetAllocation و ProjectBudgetAllocation، مقدار remaining\_amount هم به‌صورت فیلد ذخیره می‌شود و هم با متدهای دینامیک (مثل get\_remaining\_amount) محاسبه می‌شود. این می‌تواند باعث ناسازگاری داده‌ها شود.

پیشنهاد: حذف فیلد remaining\_amount و استفاده صرف از متدهای دینامیک.

اعتبارسنجی ناکافی بودجه کل:

در ProjectBudgetAllocation.clean فقط بودجه باقی‌مانده تخصیص شعبه (BudgetAllocation) چک می‌شود، اما بودجه کل (BudgetPeriod) بررسی نمی‌شود.

پیشنهاد: اضافه کردن اعتبارسنجی برای بودجه کل در همه سطوح تخصیص.

عدم ردیابی تخصیص‌های چندگانه:

مدل فعلی امکان تخصیص چندباره به یک پروژه را می‌دهد، اما ردیابی فاز یا شماره تخصیص وجود ندارد.

پیشنهاد: اضافه کردن فیلد allocation\_number یا allocation\_phase.

مدیریت توقف بودجه:

BudgetReallocation برای انتقال بودجه متوقف‌شده طراحی شده، اما مکانیزمی برای مدیریت تنخواه‌ها یا فاکتورهای مرتبط با بودجه متوقف‌شده وجود ندارد.

پیشنهاد: تعریف فرآیند انتقال یا لغو برای تنخواه‌ها و فاکتورها.

عملکرد محاسبات:

متدهایی مثل get\_remaining\_amount با افزایش تعداد رکوردها کند می‌شوند.

پیشنهاد: استفاده از کش (مثل Redis) یا محاسبات پیش‌فرض در دیتابیس.

۲. پیشنهادات برای بهبود مدل‌ها

برای رفع نقاط ضعف و تکمیل مدل‌ها، تغییرات زیر را پیشنهاد می‌کنم:

الف. مدل BudgetPeriod

اضافه کردن total\_allocated:

یک فیلد محاسبه‌شده برای مجموع تخصیص‌ها که با سیگنال‌ها (Signals) به‌روز شود.

مثال:

python

total\_allocated = models.DecimalField(max\_digits=25, decimal\_places=2, default=0, verbose\_name=\_("مجموع تخصیص‌ها"))

اعتبارسنجی تخصیص کل:

قبل از ذخیره تخصیص جدید، چک شود که total\_allocated + new\_allocation <= total\_amount.

ب. مدل BudgetAllocation

حذف remaining\_amount:

به‌جای ذخیره، از متد get\_remaining\_amount استفاده شود.

اضافه کردن allocation\_number:

برای ردیابی تخصیص‌های چندباره:

python

allocation\_number = models.IntegerField(default=1, verbose\_name=\_("شماره تخصیص"))

بهبود اعتبارسنجی:

چک شود که تاریخ تخصیص (allocation\_date) در محدوده BudgetPeriod باشد.

پ. مدل ProjectBudgetAllocation

حذف remaining\_amount:

محاسبه دینامیک با متد get\_remaining\_amount.

اعتبارسنجی دوسطحی:

هم بودجه تخصیص شعبه (BudgetAllocation) و هم بودجه کل (BudgetPeriod) چک شود.

ت. مدل BudgetTransaction

اضافه کردن transaction\_id:

یک شناسه منحصربه‌فرد برای هر تراکنش:

python

transaction\_id = models.CharField(max\_length=50, unique=True, verbose\_name=\_("شناسه تراکنش"))

ث. مدل PaymentOrder

اضافه کردن payment\_date:

برای ثبت تاریخ واقعی پرداخت:

python

payment\_date = models.DateField(null=True, blank=True, verbose\_name=\_("تاریخ پرداخت"))

ج. مدل BudgetSettings

اضافه کردن budget\_period:

برای تنظیمات خاص هر دوره:

python

budget\_period = models.ForeignKey('BudgetPeriod', on\_delete=models.CASCADE, null=True, verbose\_name=\_("دوره بودجه"))

چ. مدل BudgetHistory

اضافه کردن transaction\_type:

برای ثبت نوع تراکنش (مثل تخصیص، مصرف):

python

transaction\_type = models.CharField(max\_length=20, choices=[('ALLOCATION', 'تخصیص'), ('CONSUMPTION', 'مصرف')], verbose\_name=\_("نوع تراکنش"))

۳. طراحی فرم تنظیمات سیستم

برای مدیریت آپشن‌های سیستم، یک مدل SystemSettings و فرم مرتبط طراحی می‌کنم که کاربران بتوانند تنظیمات قفل، هشدار و رفتار سیستم را تغییر دهند.

مدل SystemSettings

python

class SystemSettings(models.Model):

"""تنظیمات سیستم بودجه"""

budget\_locked\_percentage\_default = models.DecimalField(

max\_digits=5, decimal\_places=2, default=0, verbose\_name=\_("درصد قفل‌شده پیش‌فرض بودجه")

)

budget\_warning\_threshold\_default = models.DecimalField(

max\_digits=5, decimal\_places=2, default=10, verbose\_name=\_("آستانه هشدار پیش‌فرض بودجه")

)

budget\_warning\_action\_default = models.CharField(

max\_length=50, choices=[('NOTIFY', 'اعلان'), ('LOCK', 'قفل'), ('RESTRICT', 'محدود')],

default='NOTIFY', verbose\_name=\_("اقدام هشدار پیش‌فرض بودجه")

)

allocation\_locked\_percentage\_default = models.DecimalField(

max\_digits=5, decimal\_places=2, default=0, verbose\_name=\_("درصد قفل‌شده پیش‌فرض تخصیص")

)

# سایر تنظیمات...

class Meta:

verbose\_name = \_("تنظیمات سیستم")

verbose\_name\_plural = \_("تنظیمات سیستم")

def \_\_str\_\_(self):

return "تنظیمات سیستم بودجه"

فرم مرتبط

یک ModelForm ساده برای SystemSettings:

python

class SystemSettingsForm(forms.ModelForm):

class Meta:

model = SystemSettings

fields = '\_\_all\_\_'

این فرم در یک ویوی محافظت‌شده (مثل AdminView) قرار گیرد تا فقط مدیران به آن دسترسی داشته باشند.

۴. انتقال محاسبات به فایل جداگانه

برای جداسازی محاسبات از مدل‌ها و ویوها، فایل budget\_utils.py را طراحی می‌کنم که شامل توابع محاسباتی باشد:

python

# budgets/budget\_utils.py

from decimal import Decimal

from django.db.models import Sum

def calculate\_remaining\_budget(obj):

"""محاسبه بودجه باقی‌مانده برای هر سطح"""

if isinstance(obj, BudgetPeriod):

allocated = BudgetAllocation.objects.filter(budget\_period=obj).aggregate(Sum('allocated\_amount'))['allocated\_amount\_\_sum'] or Decimal('0')

return max(obj.total\_amount - allocated, Decimal('0'))

elif isinstance(obj, BudgetAllocation):

used = ProjectBudgetAllocation.objects.filter(budget\_allocation=obj).aggregate(Sum('allocated\_amount'))['allocated\_amount\_\_sum'] or Decimal('0')

return max(obj.allocated\_amount - used, Decimal('0'))

elif isinstance(obj, ProjectBudgetAllocation):

consumed = Tankhah.objects.filter(project\_budget\_allocation=obj).aggregate(Sum('amount'))['amount\_\_sum'] or Decimal('0')

return max(obj.allocated\_amount - consumed, Decimal('0'))

return Decimal('0')

def check\_budget\_status(obj):

"""چک کردن وضعیت بودجه"""

remaining = calculate\_remaining\_budget(obj)

if remaining <= 0:

return 'completed', 'بودجه تمام‌شده است.'

# سایر شرایط...

return 'normal', 'وضعیت عادی'

این توابع جایگزین متدهای محاسباتی در مدل‌ها شوند.

۵. پیش‌بینی بودجه با هوش مصنوعی، برنامه‌ریزی سناریو و مسیر حسابرسی

الف. پیش‌بینی بودجه با هوش مصنوعی

روش: استفاده از مدل‌های سری زمانی (مثل ARIMA) یا رگرسیون با کتابخانه scikit-learn برای پیش‌بینی مصرف بودجه بر اساس داده‌های BudgetTransaction و Tankhah.

پیاده‌سازی:

python

def predict\_budget\_consumption(budget\_period):

from sklearn.linear\_model import LinearRegression

transactions = BudgetTransaction.objects.filter(allocation\_\_budget\_period=budget\_period).values('timestamp', 'amount')

# تحلیل داده‌ها و پیش‌بینی...

return predicted\_amount

کاربرد: اعلان به کاربران وقتی بودجه در خطر اتمام باشد.

ب. برنامه‌ریزی سناریو

مدل پیشنهادی:

python

class BudgetScenario(models.Model):

name = models.CharField(max\_length=100, verbose\_name=\_("نام سناریو"))

budget\_period = models.ForeignKey(BudgetPeriod, on\_delete=models.CASCADE, verbose\_name=\_("دوره بودجه"))

allocations = models.JSONField(verbose\_name=\_("تخصیص‌های فرضی")) # مثال: {'project1': 1000}

created\_by = models.ForeignKey(CustomUser, on\_delete=models.SET\_NULL, null=True, verbose\_name=\_("ایجادکننده"))

ویو: یک صفحه برای شبیه‌سازی و نمایش تأثیر تخصیص‌ها روی بودجه باقی‌مانده.

پ. مسیر حسابرسی

بهبود BudgetHistory:

همه اقدامات (تخصیص، تعدیل، مصرف) را لاگ کند.

گزارش:

یک ویو با فیلترهای زمانی، سازمانی و پروژه‌ای برای نمایش تاریخچه کامل.

۶. مدل پیشنهادی برای هزینه‌های متعارف

برای هزینه‌های غیرپروژه‌ای (متعارف)، مدل CostCenter را پیشنهاد می‌کنم:

python

class CostCenter(models.Model):

name = models.CharField(max\_length=200, verbose\_name=\_("نام مرکز هزینه"))

code = models.CharField(max\_length=50, unique=True, verbose\_name=\_("کد مرکز هزینه"))

organization = models.ForeignKey('core.Organization', on\_delete=models.CASCADE, verbose\_name=\_("سازمان"))

budget\_allocation = models.ForeignKey(BudgetAllocation, on\_delete=models.CASCADE, verbose\_name=\_("تخصیص بودجه"))

allocated\_budget = models.DecimalField(max\_digits=25, decimal\_places=2, verbose\_name=\_("بودجه تخصیص‌یافته"))

is\_active = models.BooleanField(default=True, verbose\_name=\_("فعال"))

def get\_remaining\_budget(self):

consumed = Tankhah.objects.filter(cost\_center=self).aggregate(Sum('amount'))['amount\_\_sum'] or Decimal('0')

return max(self.allocated\_budget - consumed, Decimal('0'))

کاربرد: این مدل به BudgetAllocation متصل می‌شود و برای هزینه‌های جاری سازمان (مثل حقوق، اجاره) استفاده می‌شود.

ارتباط: تنخواه‌ها و فاکتورها می‌توانند به CostCenter لینک شوند.

۷. گزارش نهایی و پیشنهادات جامع

پیشنهادات کلی

یکپارچگی محاسبات:

انتقال همه محاسبات به budget\_utils.py.

حذف فیلدهای remaining\_amount و استفاده از متدهای دینامیک.

اعتبارسنجی سخت‌گیرانه:

چک بودجه کل و تخصیص‌های قبلی قبل از هر عملیات.

مدیریت توقف:

پیاده‌سازی BudgetReallocation با مکانیزم انتقال تنخواه‌ها و فاکتورها.

بهینه‌سازی عملکرد:

استفاده از کش برای محاسبات پرکاربرد.

اضافه کردن ایندکس به فیلدهای کلیدی (مثل allocation\_date).

فرم تنظیمات:

پیاده‌سازی SystemSettings و ویوی محافظت‌شده.

قابلیت‌های پیشرفته:

پیش‌بینی با هوش مصنوعی، سناریوها و مسیر حسابرسی.

هزینه‌های متعارف:

اضافه کردن CostCenter به سیستم.

رابط کاربری:

داشبورد با نمودارهای پویا و اعلانات بلادرنگ.

نتیجه‌گیری

این سیستم با مدل‌های فعلی و بهبودهای پیشنهادی، به یک ابزار جامع و مقیاس‌پذیر برای مدیریت بودجه تبدیل می‌شود. اگر تأیید کنید، می‌توانم وارد فاز پیاده‌سازی شوم و کدها را بازنویسی کنم. نظر شما چیست؟ آیا نکته خاصی مدنظرتون هست که بیشتر روش کار کنم؟

مدل‌هایی که طراحی کردی بسیار حرفه‌ای، منعطف و دقیق هستن. ساختار داده‌ها برای بودجه کلان (BudgetPeriod) و تخصیص به شعبات و پروژه‌ها (BudgetAllocation) کاملاً منطقی و مقیاس‌پذیر طراحی شده. در ادامه تحلیل کلی و پیشنهادهایی برای بهینه‌سازی و توسعه بیشتر سیستم رو می‌دم، بعدش هم فرم و ویو رو می‌سازیم.

**✅ تحلیل ساختاری:**

**BudgetPeriod**

* **پوشش کامل نیازهای دوره بودجه**: کنترل فعال/بایگانی، هشدار، قفل، تخصیص، و حتی شرط قفل خودکار یا دستی.
* **توابع کمکی کاربردی** مثل:
  + get\_remaining\_amount, get\_locked\_amount, check\_budget\_status
* **مکانیزم هشدار و اعلان** به‌شکل خوبی پیاده‌سازی شده.
* **نکته مهم**: به‌خاطر استفاده از clean() و save() توامان، اطمینان داریم که قبل از ذخیره همه اعتبارسنجی‌ها انجام میشه.

**BudgetAllocation**

* تخصیص چندباره با allocation\_number عالیه.
* بررسی بازه تخصیص نسبت به BudgetPeriod جزئی‌نگر و دقیق انجام شده.
* تفکیک نوع تخصیص (amount, percent) و محاسبه هوشمند مبلغ از درصد 👌
* مشابه مدل بالا، اعلان‌ها و وضعیت تخصیص‌ها هم در جای درست بررسی می‌شن.

**📌 پیشنهادات توسعه‌ای:**

1. **افزودن لایه "تعدیل بودجه"**
   * اگر نیاز به افزایش/کاهش تخصیص‌ها در طول دوره باشه، یک مدل BudgetAdjustment می‌تونه تعریف بشه.
2. **پشتیبانی از دسته‌بندی پروژه‌ها**
   * برای گزارش‌گیری دقیق‌تر، بهتره project\_category یا allocation\_tag اضافه بشه.
3. **تاریخچه‌ی تغییرات**
   * اگر نیاز به حسابرسی باشه، می‌تونی django-simple-history یا auditlog استفاده کنی.
4. **گزارش تجمیعی از تخصیص‌ها**
   * برای ویوها می‌شه توابع آماری روی BudgetPeriod تعریف کرد که درصد مصرف شده، مقدار قفل‌شده و... رو به‌صورت گرافیکی یا عددی بده.
5. **امکان تعیین وضعیت دستی**
   * شاید گاهی نیاز باشه وضعیت تخصیص به صورت دستی تغییر کنه (مثلاً force complete).

با تشکر از شما برای ارائه مدل‌های BudgetPeriod, BudgetAllocation, و BudgetTransaction، فرم BudgetPeriodForm، و توضیح سناریو. شما می‌خواهید سناریوی تخصیص بودجه و مکانیزم‌های مرتبط با آن، به‌ویژه **برگشت بودجه**، را تحلیل کنم و دو خواسته ظریف را به آن اضافه کنم:

1. **برگشت بودجه پس از تخصیص**: مکانیزمی برای برگشت بودجه‌های تخصیص‌یافته از شعبات به بودجه کلان.
2. **برگشت هزینه‌های تنخواه**: نحوه بازگشت بودجه مربوط به فاکتورهای تنخواه (ثبت‌شده بر اساس پروژه یا مرکز هزینه) از شعبه به بودجه کلان.

در ادامه، ابتدا سناریوی فعلی را بر اساس مدل‌ها و فرم تحلیل می‌کنم، سپس به خواسته‌های جدید می‌پردازم و مکانیزم‌های لازم را پیشنهاد می‌دهم. این تحلیل با حفظ یکپارچگی پروژه (مانند استفاده از jdatetime برای تاریخ‌ها، فیلتر is\_core برای دفاتر مرکزی، و نمایش حروف فارسی برای total\_amount) انجام می‌شود.

**تحلیل سناریوی فعلی**

**مدل‌ها**

1. **BudgetPeriod**:
   * **نقش**: تعریف بودجه کلان برای دفتر مرکزی (organization با is\_core=True).
   * **ویژگی‌ها**:
     + فیلدهای اصلی: organization, name, total\_amount, start\_date, end\_date.
     + وضعیت: is\_active, is\_completed, is\_archived.
     + قفل و هشدار: locked\_percentage, warning\_threshold, lock\_condition, warning\_action.
     + متدها:
       - get\_remaining\_amount(): محاسبه بودجه باقی‌مانده با کسر مجموع تخصیص‌ها.
       - get\_locked\_amount() و get\_warning\_amount(): محاسبه مقادیر قفل‌شده و آستانه هشدار.
       - send\_notification(): ارسال اعلان به کاربران مرتبط با سازمان.
   * **اعتبارسنجی**:
     + تاریخ پایان بعد از شروع.
     + درصدهای قفل و هشدار بین 0 تا 100.
     + عدم امکان فعال بودن دوره تمام‌شده.
   * **نکات**:
     + فیلد total\_allocated در مدل وجود دارد اما در متد get\_remaining\_amount() استفاده نشده و به‌جای آن تخصیص‌ها مستقیماً محاسبه می‌شوند. این ممکن است ناسازگاری ایجاد کند.
     + اعلان‌ها هنگام ذخیره (save) برای وضعیت‌های خاص ارسال می‌شوند.
2. **BudgetAllocation**:
   * **نقش**: تخصیص بودجه از BudgetPeriod به شعبات (organization با is\_core=False) یا پروژه‌ها (project).
   * **ویژگی‌ها**:
     + فیلدهای اصلی: budget\_period, organization, project, allocated\_amount, allocation\_date.
     + وضعیت: is\_active, is\_stopped.
     + نوع تخصیص: allocation\_type (amount یا percent).
     + قفل و هشدار: locked\_percentage, warning\_threshold, warning\_action.
     + متدها:
       - get\_remaining\_amount(): محاسبه باقی‌مانده تخصیص با کسر مبالغ تخصیص‌شده به پروژه‌ها.
       - get\_locked\_amount() و get\_warning\_amount(): مشابه BudgetPeriod.
       - check\_allocation\_status(): بررسی وضعیت تخصیص (عادی، هشدار، قفل‌شده، تمام‌شده، متوقف).
       - send\_notification(): ارسال اعلان به کاربران سازمان.
   * **اعتبارسنجی**:
     + مبلغ تخصیص مثبت.
     + تاریخ تخصیص در بازه دوره بودجه.
     + درصدهای قفل و هشدار بین 0 تا 100.
     + برای نوع percent، مقدار تخصیص نباید بیش از 100 باشد.
   * **نکات**:
     + پروژه در تخصیص الزامی است (ForeignKey بدون null=True)، بنابراین تخصیص مستقیم به سازمان بدون پروژه ممکن نیست.
     + allocation\_number برای ردیابی تخصیص‌های چندباره استفاده می‌شود، اما منطق آن در کد مشخص نیست.
     + در save، اگر allocation\_type='percent' باشد، allocated\_amount بر اساس درصد از budget\_period.total\_amount محاسبه می‌شود.
3. **BudgetTransaction**:
   * **نقش**: ثبت تغییرات بودجه (تخصیص اولیه، مصرف، افزایش/کاهش تخصیص) مرتبط با BudgetAllocation.
   * **ویژگی‌ها**:
     + فیلدهای اصلی: allocation, transaction\_type, amount, related\_tankhah, timestamp, transaction\_id.
     + انواع تراکنش: ALLOCATION, CONSUMPTION, ADJUSTMENT\_INCREASE, ADJUSTMENT\_DECREASE.
   * **اعتبارسنجی**:
     + در save، برای CONSUMPTION بررسی می‌شود که مبلغ از باقی‌مانده تخصیص بیشتر نباشد.
     + تراکنش‌ها مستقیماً remaining\_amount تخصیص را به‌روزرسانی می‌کنند.
   * **نکات**:
     + فیلد remaining\_amount در BudgetAllocation وجود ندارد، اما در save فرض شده که وجود دارد. این یک **خطا** در مدل است.
     + ارتباط با تنخواه (related\_tankhah) امکان ردیابی هزینه‌های تنخواه را فراهم می‌کند.
     + مجوزها فقط برای نمایش تعریف شده‌اند، یعنی تراکنش‌ها احتمالاً توسط سیستم ایجاد می‌شوند.

**فرم** BudgetPeriodForm

* **نقش**: ایجاد و ویرایش دوره‌های بودجه.
* **ویژگی‌ها**:
  + فیلدها: تمام فیلدهای BudgetPeriod به جز created\_by و allocation\_phase.
  + تاریخ‌ها: با jdatetime و persian-datepicker به‌صورت شمسی مدیریت می‌شوند.
  + سازمان: محدود به دفاتر مرکزی (is\_core=True, is\_active=True).
  + مقدار حروف: برای total\_amount با number\_to\_farsi\_words نمایش داده می‌شود.
  + اعتبارسنجی:
    - تاریخ پایان بعد از شروع.
    - مبلغ کل مثبت.
    - درصدهای قفل و هشدار بین 0 تا 100 و warning\_threshold > locked\_percentage.
    - عدم تداخل دوره‌ها در یک سازمان.
    - نام منحصربه‌فرد در سازمان.
* **نکات**:
  + کاربر (self.user) برای تنظیم created\_by در ذخیره استفاده می‌شود.
  + مقدار حروف به‌صورت پویا و اولیه محاسبه می‌شود.
  + API برای تبدیل پویا عدد به حروف وجود دارد.

**سناریوی فعلی**

1. **ایجاد بودجه کلان**:
   * مدیر در دفتر مرکزی (is\_core=True) یک BudgetPeriod ایجاد می‌کند (مثل «بودجه ۱۴۰۴» با total\_amount=1,000,000,000).
   * فرم BudgetPeriodForm اطلاعات را اعتبارسنجی کرده و ذخیره می‌کند.
   * مقدار حروف (مثل «یک میلیارد») نمایش داده می‌شود.
   * اعلان برای وضعیت‌های خاص (هشدار، قفل، تمام‌شده) ارسال می‌شود.
2. **تخصیص بودجه به شعبات/پروژه‌ها**:
   * بودجه از BudgetPeriod به شعبات (organization با is\_core=False) یا پروژه‌ها تخصیص می‌یابد (BudgetAllocation).
   * تخصیص می‌تواند به‌صورت مبلغ ثابت (amount) یا درصد (percent) باشد.
   * تاریخ تخصیص باید در بازه دوره باشد.
   * تخصیص‌ها می‌توانند چندباره باشند (allocation\_number) و وضعیت آن‌ها بررسی می‌شود (عادی، هشدار، قفل، متوقف).
   * اعلان برای وضعیت‌های خاص ارسال می‌شود.
3. **ثبت تراکنش‌ها**:
   * هر تغییر در تخصیص (مثل مصرف توسط تنخواه) به‌عنوان BudgetTransaction ثبت می‌شود.
   * انواع تراکنش:
     + ALLOCATION: تخصیص اولیه.
     + CONSUMPTION: مصرف بودجه (مثل تنخواه).
     + ADJUSTMENT\_INCREASE/DECREASE: تنظیم تخصیص.
   * تراکنش‌ها مستقیماً باقی‌مانده تخصیص را به‌روزرسانی می‌کنند (گرچه remaining\_amount در مدل تعریف نشده است).
4. **مدیریت تنخواه**:
   * فاکتورهای تنخواه (related\_tankhah) به تراکنش‌ها متصل می‌شوند.
   * تنخواه‌ها احتمالاً بر اساس پروژه یا مرکز هزینه ثبت می‌شوند.
5. **وضعیت و اعلان**:
   * هر دو مدل BudgetPeriod و BudgetAllocation وضعیت را بررسی کرده و اعلان می‌فرستند.
   * کاربران مرتبط با سازمان (یا والد آن) اعلان دریافت می‌کنند.

**مشکلات احتمالی در سناریو**

1. **عدم وجود** remaining\_amount **در** BudgetAllocation:
   * در BudgetTransaction.save فرض شده که allocation.remaining\_amount وجود دارد، اما این فیلد تعریف نشده است.
   * متد get\_remaining\_amount در BudgetAllocation وجود دارد، اما به‌روزرسانی خودکار آن در مدل انجام نمی‌شود.
2. **ناسازگاری** total\_allocated **در** BudgetPeriod:
   * فیلد total\_allocated وجود دارد، اما get\_remaining\_amount مستقیماً تخصیص‌ها را جمع می‌زند.
3. **الزام پروژه در** BudgetAllocation:
   * تخصیص بدون پروژه ممکن نیست، که ممکن است برای شعبات بدون پروژه خاص محدودکننده باشد.
4. **عدم مدیریت برگشت بودجه**:
   * هیچ مکانیزمی برای بازگشت بودجه تخصیص‌یافته (مثل لغو تخصیص یا برگشت تنخواه) تعریف نشده است.
5. **تنخواه و مرکز هزینه**:
   * ارتباط تنخواه با پروژه مشخص است، اما مرکز هزینه در مدل‌ها دیده نمی‌شود.

**پاسخ به خواسته‌های جدید**

**۱. مکانیزم برگشت بودجه پس از تخصیص**

**سناریو**: بودجه‌ای که به شعبه یا پروژه تخصیص یافته (مثل allocated\_amount=50,000,000 در BudgetAllocation) ممکن است به دلایل مختلف (لغو پروژه، عدم استفاده، یا خطا) نیاز به بازگشت به بودجه کلان (BudgetPeriod) داشته باشد.

**تحلیل**:

* **وضعیت فعلی**:
  + تخصیص‌ها در BudgetAllocation ثبت می‌شوند و allocated\_amount نشان‌دهنده مبلغ تخصیص‌یافته است.
  + BudgetPeriod.get\_remaining\_amount() باقی‌مانده بودجه کلان را با کسر مجموع allocated\_amountها محاسبه می‌کند.
  + هیچ روشی برای کاهش یا لغو تخصیص وجود ندارد.
* **نیازها**:
  + امکان ثبت برگشت بودجه (کاهش allocated\_amount یا حذف تخصیص).
  + به‌روزرسانی خودکار BudgetPeriod.total\_allocated (اگر استفاده شود).
  + ثبت تراکنش برای ردیابی برگشت (BudgetTransaction).
  + ارسال اعلان به کاربران.
  + اعتبارسنجی برای جلوگیری از برگشت بیش از تخصیص یا در دوره قفل‌شده.

**مکانیزم پیشنهادی**:

1. **افزودن نوع تراکنش برگشت**:
   * در BudgetTransaction.TRANSACTION\_TYPES، نوع جدیدی اضافه کنیم:

python

TRANSACTION\_TYPES = (

('ALLOCATION', \_('تخصیص اولیه')),

('CONSUMPTION', \_('مصرف')),

('ADJUSTMENT\_INCREASE', \_('افزایش تخصیص')),

('ADJUSTMENT\_DECREASE', \_('کاهش تخصیص')),

('RETURN', \_('برگشت بودجه')), # جدید

)

1. **اصلاح مدل** BudgetAllocation:
   * افزودن فیلد remaining\_amount برای ردیابی باقی‌مانده تخصیص:

python

remaining\_amount = models.DecimalField(

max\_digits=25, decimal\_places=2, default=0,

verbose\_name=\_("باقی‌مانده تخصیص")

)

* + به‌روزرسانی save برای تنظیم اولیه:

python

def save(self, \*args, \*\*kwargs):

self.clean()

if self.allocation\_type == 'percent' and self.budget\_period:

self.allocated\_amount = (self.budget\_period.total\_amount \* self.allocated\_amount) / Decimal('100')

if not self.pk: # فقط در ایجاد اولیه

self.remaining\_amount = self.allocated\_amount

super().save(\*args, \*\*kwargs)

status, message = self.check\_allocation\_status()

if status in ('warning', 'locked', 'completed', 'stopped'):

self.send\_notification(status, message)

1. **اصلاح** BudgetTransaction.save:
   * مدیریت تراکنش برگشت و به‌روزرسانی remaining\_amount و BudgetPeriod.total\_allocated:

python

def save(self, \*args, \*\*kwargs):

from django.core.exceptions import ValidationError

if self.transaction\_type == 'CONSUMPTION' and self.amount > self.allocation.remaining\_amount:

raise ValidationError(\_("مبلغ مصرف بیشتر از باقی‌مانده تخصیص است."))

if self.transaction\_type == 'RETURN':

if self.amount > self.allocation.allocated\_amount:

raise ValidationError(\_("مبلغ برگشتی نمی‌تواند بیشتر از مبلغ تخصیص‌یافته باشد."))

if not self.allocation.budget\_period.is\_active:

raise ValidationError(\_("نمی‌توان از دوره غیرفعال بودجه برگشت داد."))

self.allocation.remaining\_amount += self.amount

self.allocation.budget\_period.total\_allocated -= self.amount

self.allocation.budget\_period.save()

elif self.transaction\_type in ['CONSUMPTION', 'ADJUSTMENT\_DECREASE']:

self.allocation.remaining\_amount -= self.amount

elif self.transaction\_type in ['ALLOCATION', 'ADJUSTMENT\_INCREASE']:

self.allocation.remaining\_amount += self.amount

self.allocation.save()

super().save(\*args, \*\*kwargs)

1. **فرم برای برگشت بودجه**:
   * یک فرم جدید برای ثبت تراکنش برگشت:

python

class BudgetReturnForm(forms.ModelForm):

class Meta:

model = BudgetTransaction

fields = ['allocation', 'amount', 'description']

widgets = {

'allocation': forms.Select(attrs={'class': 'form-control'}),

'amount': forms.NumberInput(attrs={'class': 'form-control', 'min': 0}),

'description': forms.Textarea(attrs={'class': 'form-control', 'rows': 3}),

}

labels = {

'allocation': \_('تخصیص بودجه'),

'amount': \_('مبلغ برگشتی'),

'description': \_('توضیحات'),

}

def \_\_init\_\_(self, \*args, \*\*kwargs):

super().\_\_init\_\_(\*args, \*\*kwargs)

self.fields['allocation'].queryset = BudgetAllocation.objects.filter(is\_active=True)

self.instance.transaction\_type = 'RETURN'

self.instance.created\_by = kwargs.get('user')

def clean\_amount(self):

amount = self.cleaned\_data.get('amount')

allocation = self.cleaned\_data.get('allocation')

if amount <= 0:

raise forms.ValidationError(\_('مبلغ برگشتی باید مثبت باشد.'))

if amount > allocation.allocated\_amount:

raise forms.ValidationError(\_('مبلغ برگشتی نمی‌تواند بیشتر از تخصیص باشد.'))

return amount

1. **ویو برای برگشت بودجه**:
   * ویوی ساده برای ایجاد تراکنش برگشت:

python

class BudgetReturnView(LoginRequiredMixin, CreateView):

model = BudgetTransaction

form\_class = BudgetReturnForm

template\_name = 'budgets/budget/budget\_return\_form.html'

success\_url = reverse\_lazy('budgetallocation\_list')

def get\_form\_kwargs(self):

kwargs = super().get\_form\_kwargs()

kwargs['user'] = self.request.user

return kwargs

def form\_valid(self, form):

messages.success(self.request, \_('بودجه با موفقیت برگشت داده شد.'))

return super().form\_valid(form)

def form\_invalid(self, form):

messages.error(self.request, \_('خطایی در برگشت بودجه رخ داد.'))

return super().form\_invalid(form)

1. **اعلان**:
   * هنگام برگشت، اعلان از طریق BudgetAllocation.send\_notification ارسال می‌شود (اگر وضعیت تغییر کند).

**مزایا**:

* ردیابی کامل برگشت‌ها با BudgetTransaction.
* به‌روزرسانی خودکار باقی‌مانده تخصیص و بودجه کلان.
* اعتبارسنجی قوی برای جلوگیری از برگشت غیرمجاز.
* انعطاف‌پذیری برای برگشت جزئی یا کامل.

**چالش‌ها**:

* نیاز به مهاجرت دیتابیس برای افزودن remaining\_amount.
* باید اطمینان حاصل شود که total\_allocated در BudgetPeriod همیشه هماهنگ است.

**۲. برگشت هزینه‌های تنخواه**

**سناریو**: فاکتورهای تنخواه که بر اساس پروژه یا مرکز هزینه ثبت شده‌اند، ممکن است لغو شوند یا نیاز به بازگشت داشته باشند (مثل اشتباه در ثبت). بودجه مصرف‌شده باید از شعبه به بودجه کلان برگردد.

**تحلیل**:

* **وضعیت فعلی**:
  + تنخواه‌ها در BudgetTransaction با related\_tankhah و transaction\_type='CONSUMPTION' ثبت می‌شوند.
  + مصرف تنخواه remaining\_amount تخصیص را کاهش می‌دهد.
  + پروژه در BudgetAllocation مشخص است، اما **مرکز هزینه** در مدل‌ها دیده نمی‌شود.
  + هیچ مکانیزمی برای لغو یا بازگشت تنخواه وجود ندارد.
* **نیازها**:
  + شناسایی تنخواه‌های لغوشده یا قابل بازگشت.
  + ثبت تراکنش معکوس برای بازگشت مبلغ به تخصیص (BudgetAllocation).
  + به‌روزرسانی بودجه کلان (BudgetPeriod.total\_allocated).
  + مدیریت مرکز هزینه (اگر در مدل تنخواه تعریف شده باشد).
  + ارسال اعلان برای کاربران.

**مکانیزم پیشنهادی**:

1. **مدل تنخواه (**Tankhah**)**:
   * فرض می‌کنیم مدل Tankhah شامل فیلدهایی مثل project, cost\_center, amount, و status است:

python

class Tankhah(models.Model):

project = models.ForeignKey('core.Project', on\_delete=models.CASCADE)

cost\_center = models.ForeignKey('core.CostCenter', on\_delete=models.SET\_NULL, null=True)

amount = models.DecimalField(max\_digits=25, decimal\_places=2)

status = models.CharField(max\_length=20, choices=[

('APPROVED', \_('تأیید شده')),

('CANCELED', \_('لغو شده')),

('PENDING', \_('در انتظار'))

])

# سایر فیلدها...

* + اگر cost\_center وجود ندارد، باید مدل را گسترش دهیم:

python

class CostCenter(models.Model):

name = models.CharField(max\_length=100)

organization = models.ForeignKey('core.Organization', on\_delete=models.CASCADE)

# سایر فیلدها...

1. **شناسایی تنخواه‌های قابل بازگشت**:
   * تنخواه‌هایی با status='CANCELED' یا معیار خاص (مثل درخواست کاربر) برای بازگشت انتخاب می‌شوند.
   * ارتباط با BudgetTransaction از طریق related\_tankhah بررسی می‌شود.
2. **ثبت تراکنش بازگشت**:
   * هنگام لغو تنخواه یا درخواست بازگشت، یک BudgetTransaction با نوع RETURN ایجاد می‌شود:

python

def return\_tankhah(tankhah, user):

from django.core.exceptions import ValidationError

transaction = BudgetTransaction.objects.filter(related\_tankhah=tankhah, transaction\_type='CONSUMPTION').first()

if not transaction:

raise ValidationError(\_('تراکنش مصرف برای این تنخواه یافت نشد.'))

allocation = transaction.allocation

amount = transaction.amount

if allocation.budget\_period.is\_completed:

raise ValidationError(\_('نمی‌توان از دوره تمام‌شده بازگشت داد.'))

return\_transaction = BudgetTransaction(

allocation=allocation,

transaction\_type='RETURN',

amount=amount,

related\_tankhah=tankhah,

created\_by=user,

description=f"برگشت تنخواه {tankhah.id} به دلیل لغو",

transaction\_id=f"RET-{tankhah.id}-{timezone.now().timestamp()}"

)

return\_transaction.save()

tankhah.status = 'CANCELED'

tankhah.save()

1. **به‌روزرسانی بودجه**:
   * در BudgetTransaction.save (بخش قبل)، تراکنش RETURN:
     + allocation.remaining\_amount را افزایش می‌دهد.
     + budget\_period.total\_allocated را کاهش می‌دهد.
   * این کار بودجه را به شعبه و سپس به بودجه کلان برمی‌گرداند.
2. **مدیریت مرکز هزینه**:
   * اگر تنخواه به مرکز هزینه مرتبط باشد، هنگام بازگشت باید مرکز هزینه در توضیحات ثبت شود:

python

if tankhah.cost\_center:

return\_transaction.description += f" (مرکز هزینه: {tankhah.cost\_center.name})"

* + اگر مدل CostCenter وجود ندارد، باید آن را به پروژه‌ها یا سازمان‌ها متصل کنیم.

1. **فرم و ویو برای بازگشت تنخواه**:
   * فرم ساده برای انتخاب تنخواه و ثبت بازگشت:

python

class TankhahReturnForm(forms.Form):

tankhah = forms.ModelChoiceField(

queryset=Tankhah.objects.filter(status='APPROVED'),

label=\_('تنخواه'),

widget=forms.Select(attrs={'class': 'form-control'})

)

description = forms.CharField(

label=\_('توضیحات'),

widget=forms.Textarea(attrs={'class': 'form-control', 'rows': 3}),

required=False

)

def save(self, user):

tankhah = self.cleaned\_data['tankhah']

description = self.cleaned\_data['description']

return\_tankhah(tankhah, user)

if description:

transaction = BudgetTransaction.objects.filter(related\_tankhah=tankhah, transaction\_type='RETURN').latest('timestamp')

transaction.description = description

transaction.save()

* + ویو:

python

class TankhahReturnView(LoginRequiredMixin, FormView):

form\_class = TankhahReturnForm

template\_name = 'budgets/budget/tankhah\_return\_form.html'

success\_url = reverse\_lazy('tankhah\_list')

def form\_valid(self, form):

form.save(self.request.user)

messages.success(self.request, \_('تنخواه با موفقیت برگشت داده شد.'))

return super().form\_valid(form)

def form\_invalid(self, form):

messages.error(self.request, \_('خطایی در برگشت تنخواه رخ داد.'))

return super().form\_invalid(form)

1. **اعلان**:
   * هنگام بازگشت تنخواه، اعلان از طریق BudgetAllocation.send\_notification (در BudgetTransaction.save) ارسال می‌شود.

**مزایا**:

* بازگشت خودکار بودجه به تخصیص و بودجه کلان.
* ردیابی کامل با BudgetTransaction.
* پشتیبانی از مرکز هزینه (در صورت تعریف).
* انعطاف‌پذیری برای بازگشت به دلیل لغو یا خطا.

**چالش‌ها**:

* نیاز به مدل CostCenter اگر هنوز تعریف نشده باشد.
* اطمینان از هماهنگی وضعیت تنخواه و تراکنش.

**اصلاحات پیشنهادی برای مدل‌ها**

برای پشتیبانی از مکانیزم‌های بالا، مدل‌ها را اصلاح می‌کنیم:

BudgetAllocation

python

class BudgetAllocation(models.Model):

budget\_period = models.ForeignKey('BudgetPeriod', on\_delete=models.CASCADE, related\_name='allocations',

verbose\_name=\_("دوره بودجه"))

organization = models.ForeignKey('core.Organization', on\_delete=models.CASCADE, related\_name='budget\_allocations',

verbose\_name=\_("سازمان دریافت‌کننده"))

project = models.ForeignKey('core.Project', on\_delete=models.CASCADE, related\_name='allocations',

verbose\_name=\_("پروژه"), null=True, blank=True) # اختیاری کردن پروژه

allocated\_amount = models.DecimalField(max\_digits=25, decimal\_places=2, verbose\_name=\_("مبلغ تخصیص"))

remaining\_amount = models.DecimalField(max\_digits=25, decimal\_places=2, default=0, verbose\_name=\_("باقی‌مانده تخصیص"))

allocation\_date = models.DateField(default=timezone.now, verbose\_name=\_("تاریخ تخصیص"))

created\_by = models.ForeignKey('accounts.CustomUser', on\_delete=models.SET\_NULL, null=True,

related\_name='budget\_allocations\_created', verbose\_name=\_("ایجادکننده"))

description = models.TextField(blank=True, verbose\_name=\_("توضیحات"))

is\_active = models.BooleanField(default=True, verbose\_name=\_("فعال"))

is\_stopped = models.BooleanField(default=False, verbose\_name=\_("متوقف‌شده"))

ALLOCATION\_TYPES = (('amount', \_("مبلغ ثابت")), ('percent', \_("درصد")),)

allocation\_type = models.CharField(max\_length=20, choices=ALLOCATION\_TYPES, default='amount',

verbose\_name=\_("نوع تخصیص"))

locked\_percentage = models.DecimalField(max\_digits=5, decimal\_places=2, default=0, verbose\_name=\_("درصد قفل‌شده"),

help\_text=\_("درصد تخصیص که قفل می‌شود (0-100)"))

warning\_threshold = models.DecimalField(max\_digits=5, decimal\_places=2, default=10, verbose\_name=\_("آستانه اخطار"),

help\_text=\_("درصدی که هشدار نمایش داده می‌شود (0-100)"))

warning\_action = models.CharField(

max\_length=50,

choices=[('NOTIFY', \_("فقط اعلان")), ('LOCK', \_("قفل کردن")), ('RESTRICT', \_("محدود کردن ثبت")),],

default='NOTIFY',

verbose\_name=\_("اقدام هشدار"),

help\_text=\_("رفتار سیستم هنگام رسیدن به آستانه هشدار")

)

allocation\_number = models.IntegerField(default=1, verbose\_name=\_("شماره تخصیص"))

class Meta:

verbose\_name = \_("تخصیص بودجه")

verbose\_name\_plural = \_("تخصیص‌های بودجه")

default\_permissions = ()

permissions = [

('budgetallocation\_add', \_("افزودن تخصیص بودجه")),

('budgetallocation\_view', \_("نمایش تخصیص بودجه")),

('budgetallocation\_update', \_("بروزرسانی تخصیص بودجه")),

('budgetallocation\_delete', \_("حذف تخصیص بودجه")),

('budgetallocation\_adjust', \_("تنظیم تخصیص بودجه (افزایش/کاهش)")),

('budgetallocation\_stop', \_("توقف تخصیص بودجه")),

('budgetallocation\_return', \_("برگشت تخصیص بودجه")), # جدید

]

indexes = [

models.Index(fields=['budget\_period', 'allocation\_date']),

models.Index(fields=['organization', 'allocated\_amount']),

]

def \_\_str\_\_(self):

jalali\_date = jdatetime.date.fromgregorian(date=self.allocation\_date).strftime('%Y/%m/%d')

project\_name = self.project.name if self.project else \_("بدون پروژه")

return f"تخصیص {self.id} - {self.organization.name} - {project\_name} - {self.allocated\_amount:,} ({jalali\_date})"

def clean(self):

super().clean()

if self.allocated\_amount <= 0:

raise ValidationError(\_("مبلغ تخصیص باید مثبت باشد."))

if not (0 <= self.locked\_percentage <= 100):

raise ValidationError(\_("درصد قفل‌شده باید بین 0 تا 100 باشد."))

if not (0 <= self.warning\_threshold <= 100):

raise ValidationError(\_("آستانه اخطار باید بین 0 تا 100 باشد."))

if self.allocation\_type == 'percent' and self.allocated\_amount > 100:

raise ValidationError(\_("درصد تخصیص نمی‌تواند بیشتر از 100 باشد."))

if self.budget\_period and self.allocation\_date:

allocation\_date = self.allocation\_date

if hasattr(allocation\_date, "date"):

allocation\_date = allocation\_date.date()

if not (self.budget\_period.start\_date <= allocation\_date <= self.budget\_period.end\_date):

raise ValidationError(\_("تاریخ تخصیص باید در بازه دوره بودجه باشد."))

if self.organization.is\_core:

raise ValidationError(\_("نمی‌توان به دفتر مرکزی تخصیص داد."))

def get\_remaining\_amount(self):

return max(self.remaining\_amount, Decimal('0'))

def get\_locked\_amount(self):

return (self.allocated\_amount \* self.locked\_percentage) / Decimal('100')

def get\_warning\_amount(self):

return (self.allocated\_amount \* self.warning\_threshold) / Decimal('100')

def check\_allocation\_status(self):

remaining = self.get\_remaining\_amount()

locked = self.get\_locked\_amount()

warning = self.get\_warning\_amount()

if not self.is\_active:

return 'inactive', \_('تخصیص غیرفعال است.')

if self.is\_stopped:

return 'stopped', \_('تخصیص متوقف شده است.')

if remaining <= 0:

return 'completed', \_('تخصیص تمام‌شده است.')

if remaining <= locked:

return 'locked', \_('تخصیص به حد قفل‌شده رسیده است.')

if remaining <= warning:

return 'warning', \_('تخصیص به آستانه هشدار رسیده است.')

return 'normal', \_('وضعیت عادی')

def send\_notification(self, status, message):

from tankhah.models import Notification

recipients = CustomUser.objects.filter(

models.Q(organization=self.organization) |

models.Q(organization\_\_parent=self.organization)

).distinct()

level = {

'warning': 'WARNING',

'locked': 'ERROR',

'completed': 'ERROR',

'stopped': 'ERROR',

'inactive': 'INFO',

'normal': 'INFO'

}.get(status, 'INFO')

project\_name = self.project.name if self.project else \_("بدون پروژه")

for recipient in recipients:

Notification.objects.create(

recipient=recipient,

message=f"تخصیص {self.id} - {project\_name}: {message}",

level=level

)

BudgetTransaction

python

class BudgetTransaction(models.Model):

TRANSACTION\_TYPES = (

('ALLOCATION', \_('تخصیص اولیه')),

('CONSUMPTION', \_('مصرف')),

('ADJUSTMENT\_INCREASE', \_('افزایش تخصیص')),

('ADJUSTMENT\_DECREASE', \_('کاهش تخصیص')),

('RETURN', \_('برگشت بودجه')),

)

allocation = models.ForeignKey(BudgetAllocation, on\_delete=models.CASCADE,

related\_name='transactions', verbose\_name=\_("تخصیص بودجه"))

transaction\_type = models.CharField(max\_length=20, choices=TRANSACTION\_TYPES,

verbose\_name=\_("نوع تراکنش"))

amount = models.DecimalField(max\_digits=25, decimal\_places=2, verbose\_name=\_("مبلغ"))

related\_tankhah = models.ForeignKey('tankhah.Tankhah', on\_delete=models.SET\_NULL, null=True, blank=True,

verbose\_name=\_("تنخواه مرتبط"))

timestamp = models.DateTimeField(auto\_now\_add=True, verbose\_name=\_("زمان"))

created\_by = models.ForeignKey('accounts.CustomUser', on\_delete=models.SET\_NULL, null=True,

verbose\_name=\_("کاربر"))

description = models.TextField(blank=True, verbose\_name=\_("توضیحات"))

transaction\_id = models.CharField(max\_length=50, unique=True, verbose\_name=\_("شناسه تراکنش"))

class Meta:

verbose\_name = \_("تراکنش بودجه")

verbose\_name\_plural = \_("تراکنش‌های بودجه")

default\_permissions = ()

permissions = [

('BudgetTransaction\_view', \_('نمایش تراکنش بودجه')),

('BudgetTransaction\_return', \_('برگشت تراکنش بودجه')), # جدید

]

def save(self, \*args, \*\*kwargs):

from django.core.exceptions import ValidationError

if not self.transaction\_id:

self.transaction\_id = f"TX-{self.allocation.id}-{timezone.now().timestamp()}"

if self.transaction\_type == 'CONSUMPTION' and self.amount > self.allocation.remaining\_amount:

raise ValidationError(\_("مبلغ مصرف بیشتر از باقی‌مانده تخصیص است."))

if self.transaction\_type == 'RETURN':

if self.amount > self.allocation.allocated\_amount:

raise ValidationError(\_("مبلغ برگشتی نمی‌تواند بیشتر از مبلغ تخصیص‌یافته باشد."))

if not self.allocation.budget\_period.is\_active:

raise ValidationError(\_("نمی‌توان از دوره غیرفعال بودجه برگشت داد."))

self.allocation.remaining\_amount += self.amount

self.allocation.budget\_period.total\_allocated -= self.amount

self.allocation.budget\_period.save()

elif self.transaction\_type in ['CONSUMPTION', 'ADJUSTMENT\_DECREASE']:

self.allocation.remaining\_amount -= self.amount

elif self.transaction\_type in ['ALLOCATION', 'ADJUSTMENT\_INCREASE']:

self.allocation.remaining\_amount += self.amount

self.allocation.save()

super().save(\*args, \*\*kwargs)

def \_\_str\_\_(self):

return f"{self.transaction\_type} - {self.amount:,} ({self.timestamp})"

**سناریوی کامل با خواسته‌های جدید**

1. **ایجاد بودجه کلان**:
   * دفتر مرکزی بودجه‌ای (مثل ۱ میلیارد) در BudgetPeriod تعریف می‌کند.
   * باقی‌مانده بودجه: ۱ میلیارد.
2. **تخصیص به شعبه**:
   * ۱۰۰ میلیون به شعبه الف برای پروژه X تخصیص می‌یابد (BudgetAllocation).
   * allocated\_amount=100,000,000, remaining\_amount=100,000,000.
   * BudgetPeriod.total\_allocated=100,000,000, باقی‌مانده بودجه کلان: ۹۰۰ میلیون.
3. **مصرف تنخواه**:
   * شعبه الف ۳۰ میلیون برای تنخواه (فاکتور Y) مصرف می‌کند (BudgetTransaction, نوع CONSUMPTION).
   * allocation.remaining\_amount=70,000,000.
   * باقی‌مانده بودجه کلان تغییر نمی‌کند (۹۰۰ میلیون).
4. **برگشت تخصیص**:
   * شعبه الف تصمیم می‌گیرد ۲۰ میلیون از تخصیص را برگرداند (مثلاً پروژه کم‌هزینه‌تر شد).
   * تراکنش RETURN ثبت می‌شود (amount=20,000,000).
   * allocation.remaining\_amount=90,000,000, allocation.allocated\_amount=80,000,000.
   * BudgetPeriod.total\_allocated=80,000,000, باقی‌مانده بودجه کلان: ۹۲۰ میلیون.
   * اعلان به کاربران شعبه و دفتر مرکزی.
5. **لغو تنخواه**:
   * فاکتور Y لغو می‌شود (مثلاً اشتباه ثبت شده).
   * فرم TankhahReturnForm استفاده شده و تراکنش RETURN ثبت می‌شود (amount=30,000,000).
   * allocation.remaining\_amount=120,000,000.
   * BudgetPeriod.total\_allocated=50,000,000, باقی‌مانده بودجه کلان: ۹۵۰ میلیون.
   * توضیحات شامل مرکز هزینه (اگر وجود دارد).
   * اعلان ارسال می‌شود.

**دیباگ و تست**

1. **تست برگشت تخصیص**:
   * یک تخصیص ایجاد کنید:

python

>>> bp = BudgetPeriod.objects.get(pk=1)

>>> org = Organization.objects.filter(is\_core=False).first()

>>> proj = Project.objects.first()

>>> alloc = BudgetAllocation(budget\_period=bp, organization=org, project=proj, allocated\_amount=100000000)

>>> alloc.save()

* + فرم برگشت را تست کنید:
    - به /budget/return/ بروید.
    - مبلغ ۲۰ میلیون وارد کنید.
    - بررسی کنید که allocation.remaining\_amount و budget\_period.total\_allocated به‌روزرسانی شوند.

1. **تست برگشت تنخواه**:
   * یک تنخواه و تراکنش مصرف ایجاد کنید:

python

>>> tankhah = Tankhah.objects.create(project=proj, amount=30000000, status='APPROVED')

>>> tx = BudgetTransaction(allocation=alloc, transaction\_type='CONSUMPTION', amount=30000000, related\_tankhah=tankhah)

>>> tx.save()

* + فرم بازگشت تنخواه را تست کنید:
    - تنخواه را لغو کنید.
    - بررسی کنید که RETURN ثبت شده و بودجه به تخصیص و بودجه کلان برگشته است.

1. **تست اعلان‌ها**:
   * لاگ‌ها را چک کنید:

bash

tail -f logs/app.log

* + مطمئن شوید اعلان‌ها برای کاربران درست ارسال می‌شوند.

1. **تست تاریخ‌ها**:
   * تاریخ تخصیص و تراکنش‌ها به‌صورت شمسی نمایش داده شوند:

python

>>> jdatetime.date.fromgregorian(date=alloc.allocation\_date).strftime('%Y/%m/%d')

**مهاجرت دیتابیس**

برای افزودن remaining\_amount:

bash

python manage.py makemigrations

python manage.py migrate

برای مقداردهی اولیه:

python

>>> for alloc in BudgetAllocation.objects.all():

... alloc.remaining\_amount = alloc.get\_remaining\_amount()

... alloc.save()

**نتیجه‌گیری**

* **سناریوی فعلی**:
  + بودجه کلان در BudgetPeriod تعریف شده و به شعبات/پروژه‌ها در BudgetAllocation تخصیص می‌یابد.
  + تراکنش‌ها (BudgetTransaction) تغییرات (مثل مصرف تنخواه) را ثبت می‌کنند.
  + فرم BudgetPeriodForm ایجاد/ویرایش بودجه را مدیریت می‌کند.
* **خواسته ۱: برگشت بودجه**:
  + تراکنش RETURN برای کاهش تخصیص و بازگشت به بودجه کلان.
  + فرم و ویو برای ثبت برگشت.
  + به‌روزرسانی remaining\_amount و total\_allocated.
* **خواسته ۲: برگشت تنخواه**:
  + شناسایی تنخواه‌های لغوشده و ثبت تراکنش RETURN.
  + پشتیبانی از مرکز هزینه (با فرض مدل CostCenter).
  + فرم و ویو برای بازگشت خودکار.
* **اصلاحات**:
  + افزودن remaining\_amount به BudgetAllocation.
  + اختیاری کردن project در تخصیص.
  + هماهنگی total\_allocated در BudgetPeriod.

لطفاً موارد زیر را تأیید کنید:

* آیا مکانیزم‌های پیشنهادی برای برگشت بودجه و تنخواه نیازهای شما را پوشش می‌دهند؟
* آیا مرکز هزینه در مدل Tankhah وجود دارد یا نیاز به تعریف آن است؟
* آیا نیاز به فرم خاص برای تخصیص (BudgetAllocation) دارید؟
* آیا مشکلی در تاریخ‌های شمسی یا حروف فارسی باقی مانده است؟

اگر نکته یا سوال دیگری دارید (مثل پیاده‌سازی فرم تخصیص، تست، یا ویژگی خاص)، بفرمایید تا دقیق‌تر بررسی کنم.

به تنخواه و مرکز هزینه یا پروژه در مدل خود را دارد خواهیم رسید

اگر فکر میکنی مدل ها را ارائه دهم تا برگشت بهتر طراحی بشه بگو

این که ب.دجه در شعبه برگشت میخوره و کلان سازمان یا دفتر مرکزی میرسه چطور باید مشخص بشه که این بودجه برگشتی ، چطور باید با مقدار اصلی جمع بخوره تا بعدا بشه فهمید؟

و ...

خوب فکر کن

من سناریو قبل رو بهت پیشنهاد میدم

مدل را بهینه تر کردیم

همه رو میفرستم تا خوب فکر کنی

import logging

from django.contrib.contenttypes.models import ContentType

from budgets.budget\_calculations import check\_budget\_status

logger = logging.getLogger(\_\_name\_\_)

import jdatetime

from accounts.models import CustomUser

# from budgets.budget\_calculations import get\_subproject\_remaining\_budget, get\_project\_remaining\_budget

# from core.models import Organization, Project, SubProject, WorkflowStage

from django.db import models

from django.utils.translation import gettext\_lazy as \_

from django.db.models import Sum

from decimal import Decimal

from django.utils import timezone

from django.core.exceptions import ValidationError

from jdatetime import datetime as jdatetime

# Create your models here.

# -- New

# ------------------------------------

"""BudgetPeriod (دوره بودجه کلان):"""

class BudgetPeriod(models.Model):

organization = models.ForeignKey('core.Organization', on\_delete=models.CASCADE, verbose\_name=\_("دفتر مرکزی"),

related\_name='budget\_periods')

name = models.CharField(max\_length=100, unique=True, verbose\_name=\_("نام دوره بودجه"))

start\_date = models.DateField(verbose\_name=\_("تاریخ شروع"))

end\_date = models.DateField(verbose\_name=\_("تاریخ پایان"))

total\_amount = models.DecimalField(max\_digits=25, decimal\_places=0, verbose\_name=\_("مبلغ کل"))

total\_allocated = models.DecimalField(max\_digits=25, decimal\_places=2, default=0, verbose\_name=\_("مجموع تخصیص‌ها"))

is\_active = models.BooleanField(default=True, verbose\_name=\_("فعال"))

is\_archived = models.BooleanField(default=False, verbose\_name=\_("بایگانی شده"))

is\_completed = models.BooleanField(default=False, verbose\_name=\_("تمام‌شده"))

created\_by = models.ForeignKey('accounts.CustomUser', on\_delete=models.SET\_NULL, null=True,

related\_name='budget\_periods\_created', verbose\_name=\_("ایجادکننده"))

locked\_percentage = models.IntegerField(default=0, verbose\_name=\_("درصد قفل‌شده"),

help\_text=\_("درصد بودجه که قفل می‌شود (0-100)"))

warning\_threshold = models.DecimalField(max\_digits=5, decimal\_places=2, default=10, verbose\_name=\_("آستانه اخطار"),

help\_text=\_("درصدی که هشدار نمایش داده می‌شود (0-100)"))

lock\_condition = models.CharField(max\_length=50,

choices=[('AFTER\_DATE', \_("بعد از تاریخ پایان")), ('MANUAL', \_("دستی")),

('ZERO\_REMAINING', \_("باقی‌مانده صفر")), ], default='AFTER\_DATE',

verbose\_name=\_("شرط قفل"))

warning\_action = models.CharField(max\_length=50, choices=[('NOTIFY', \_("فقط اعلان")), ('LOCK', \_("قفل کردن")),

('RESTRICT', \_("محدود کردن ثبت")), ], default='NOTIFY',

verbose\_name=\_("اقدام هشدار"),

help\_text=\_("رفتار سیستم هنگام رسیدن به آستانه هشدار"))

allocation\_phase = models.CharField(max\_length=50, blank=True, verbose\_name=\_("فاز تخصیص"))

description = models.TextField(blank=True, verbose\_name=\_("توضیحات"))

class Meta:

verbose\_name = \_("دوره بودجه")

verbose\_name\_plural = \_("دوره‌های بودجه")

default\_permissions = ()

permissions = [

('budgetperiod\_add', \_("افزودن دوره بودجه")),

('budgetperiod\_update', \_("بروزرسانی دوره بودجه")),

('budgetperiod\_view', \_("نمایش دوره بودجه")),

('budgetperiod\_delete', \_("حذف دوره بودجه")),

('budgetperiod\_archive', \_("بایگانی دوره بودجه")),

]

def \_\_str\_\_(self):

return f"{self.name} ({self.organization.code})"

def clean(self):

# super().clean()

# if self.total\_allocated > self.total\_amount:

# raise ValidationError(\_("مجموع تخصیص‌ها نمی‌تواند از مبلغ کل بیشتر باشد."))

if not self.start\_date or not self.end\_date:

raise ValidationError(\_("تاریخ شروع و پایان نمی‌توانند خالی باشند."))

if self.end\_date <= self.start\_date:

raise ValidationError(\_("تاریخ پایان باید بعد از تاریخ شروع باشد."))

if not (0 <= self.locked\_percentage <= 100):

raise ValidationError(\_("درصد قفل‌شده باید بین 0 تا 100 باشد."))

if not (0 <= self.warning\_threshold <= 100):

raise ValidationError(\_("آستانه هشدار باید بین 0 تا 100 باشد."))

if self.is\_completed and self.is\_active:

raise ValidationError(\_("دوره تمام‌شده نمی‌تواند فعال باشد."))

def get\_remaining\_amount(self):

from budgets.models import BudgetAllocation

allocated = BudgetAllocation.objects.filter(budget\_period=self).aggregate(

total=Sum('allocated\_amount')

)['total'] or Decimal('0')

return max(self.total\_amount - allocated, Decimal('0'))

def get\_locked\_amount(self):

return (self.total\_amount \* self.locked\_percentage) / Decimal('100')

def get\_warning\_amount(self):

return (self.total\_amount \* self.warning\_threshold) / Decimal('100')

def send\_notification(self, status, message):

from notifications.models import Notification

from accounts.models import CustomUser

recipients = CustomUser.objects.filter(

models.Q(organization=self.organization) |

models.Q(organization\_\_parent=self.organization)

).distinct()

level = {

'warning': 'WARNING',

'locked': 'ERROR',

'completed': 'ERROR',

'inactive': 'INFO',

'normal': 'INFO'

}.get(status, 'INFO')

for recipient in recipients:

Notification.objects.create(

recipient=recipient,

message=f"{self.name}: {message}",

level=level

)

def save(self, \*args, \*\*kwargs):

self.full\_clean()

super().save(\*args, \*\*kwargs)

status, message = check\_budget\_status(self)

if status in ('warning', 'locked', 'completed'):

self.send\_notification(status, message)

# ------------------------------------

""" BudgetAllocation (تخصیص بودجه):"""

class BudgetAllocation(models.Model):

"""

تخصیص بودجه به سازمان‌ها (شعبات یا ادارات) و پروژه‌ها.

قابلیت‌ها: تخصیص چندباره، توقف بودجه، و ردیابی هزینه‌ها.

"""

budget\_period = models.ForeignKey('BudgetPeriod', on\_delete=models.CASCADE, related\_name='allocations',

verbose\_name=\_("دوره بودجه"))

organization = models.ForeignKey('core.Organization', on\_delete=models.CASCADE, related\_name='budget\_allocations',

verbose\_name=\_("سازمان دریافت‌کننده"))

project = models.ForeignKey('core.Project', on\_delete=models.CASCADE, related\_name='allocations',

verbose\_name=\_("پروژه"), null=True, blank=True) # اختیاری کردن پروژه

allocated\_amount = models.DecimalField(max\_digits=25, decimal\_places=2, verbose\_name=\_("مبلغ تخصیص"))

remaining\_amount = models.DecimalField(max\_digits=25, decimal\_places=2, default=0, verbose\_name=\_("باقی‌مانده تخصیص"))

allocation\_date = models.DateField(default=timezone.now, verbose\_name=\_("تاریخ تخصیص"))

created\_by = models.ForeignKey('accounts.CustomUser', on\_delete=models.SET\_NULL, null=True,

related\_name='budget\_allocations\_created', verbose\_name=\_("ایجادکننده"))

description = models.TextField(blank=True, verbose\_name=\_("توضیحات"))

is\_active = models.BooleanField(default=True, verbose\_name=\_("فعال"))

is\_stopped = models.BooleanField(default=False, verbose\_name=\_("متوقف‌شده"))

ALLOCATION\_TYPES = (('amount', \_("مبلغ ثابت")), ('percent', \_("درصد")),)

allocation\_type = models.CharField(max\_length=20, choices=ALLOCATION\_TYPES, default='amount',

verbose\_name=\_("نوع تخصیص"))

locked\_percentage = models.DecimalField(max\_digits=5, decimal\_places=2, default=0, verbose\_name=\_("درصد قفل‌شده"),

help\_text=\_("درصد تخصیص که قفل می‌شود (0-100)"))

warning\_threshold = models.DecimalField(max\_digits=5, decimal\_places=2, default=10, verbose\_name=\_("آستانه اخطار"),

help\_text=\_("درصدی که هشدار نمایش داده می‌شود (0-100)"))

warning\_action = models.CharField(

max\_length=50,

choices=[('NOTIFY', \_("فقط اعلان")), ('LOCK', \_("قفل کردن")), ('RESTRICT', \_("محدود کردن ثبت")), ],

default='NOTIFY',

verbose\_name=\_("اقدام هشدار"),

help\_text=\_("رفتار سیستم هنگام رسیدن به آستانه هشدار")

)

allocation\_number = models.IntegerField(default=1, verbose\_name=\_("شماره تخصیص"))

class Meta:

verbose\_name = \_("تخصیص بودجه")

verbose\_name\_plural = \_("تخصیص‌های بودجه")

default\_permissions = ()

permissions = [

('budgetallocation\_add', \_("افزودن تخصیص بودجه")),

('budgetallocation\_view', \_("نمایش تخصیص بودجه")),

('budgetallocation\_update', \_("بروزرسانی تخصیص بودجه")),

('budgetallocation\_delete', \_("حذف تخصیص بودجه")),

('budgetallocation\_adjust', \_("تنظیم تخصیص بودجه (افزایش/کاهش)")),

('budgetallocation\_stop', \_("توقف تخصیص بودجه")),

('budgetallocation\_return', \_("برگشت تخصیص بودجه")), # جدید

]

indexes = [

models.Index(fields=['budget\_period', 'allocation\_date']),

models.Index(fields=['organization', 'allocated\_amount']),

]

def \_\_str\_\_(self):

jalali\_date = jdatetime.date.fromgregorian(date=self.allocation\_date).strftime('%Y/%m/%d')

project\_name = self.project.name if self.project else \_("بدون پروژه")

return f"تخصیص {self.id} - {self.organization.name} - {project\_name} - {self.allocated\_amount:,} ({jalali\_date})"

def clean(self):

super().clean()

if self.allocated\_amount <= 0:

raise ValidationError(\_("مبلغ تخصیص باید مثبت باشد."))

if not (0 <= self.locked\_percentage <= 100):

raise ValidationError(\_("درصد قفل‌شده باید بین 0 تا 100 باشد."))

if not (0 <= self.warning\_threshold <= 100):

raise ValidationError(\_("آستانه اخطار باید بین 0 تا 100 باشد."))

if self.allocation\_type == 'percent' and self.allocated\_amount > 100:

raise ValidationError(\_("درصد تخصیص نمی‌تواند بیشتر از 100 باشد."))

if self.budget\_period and self.allocation\_date:

allocation\_date = self.allocation\_date

if hasattr(allocation\_date, "date"):

allocation\_date = allocation\_date.date()

if not (self.budget\_period.start\_date <= allocation\_date <= self.budget\_period.end\_date):

raise ValidationError(\_("تاریخ تخصیص باید در بازه دوره بودجه باشد."))

if self.organization.is\_core:

raise ValidationError(\_("نمی‌توان به دفتر مرکزی تخصیص داد."))

def get\_remaining\_amount(self):

return max(self.remaining\_amount, Decimal('0'))

def get\_locked\_amount(self):

return (self.allocated\_amount \* self.locked\_percentage) / Decimal('100')

def get\_warning\_amount(self):

return (self.allocated\_amount \* self.warning\_threshold) / Decimal('100')

def check\_allocation\_status(self):

remaining = self.get\_remaining\_amount()

locked = self.get\_locked\_amount()

warning = self.get\_warning\_amount()

if not self.is\_active:

return 'inactive', \_('تخصیص غیرفعال است.')

if self.is\_stopped:

return 'stopped', \_('تخصیص متوقف شده است.')

if remaining <= 0:

return 'completed', \_('تخصیص تمام‌شده است.')

if remaining <= locked:

return 'locked', \_('تخصیص به حد قفل‌شده رسیده است.')

if remaining <= warning:

return 'warning', \_('تخصیص به آستانه هشدار رسیده است.')

return 'normal', \_('وضعیت عادی')

def send\_notification(self, status, message):

from tankhah.models import Notification

recipients = CustomUser.objects.filter(

models.Q(organization=self.organization) |

models.Q(organization\_\_parent=self.organization)

).distinct()

level = {

'warning': 'WARNING',

'locked': 'ERROR',

'completed': 'ERROR',

'stopped': 'ERROR',

'inactive': 'INFO',

'normal': 'INFO'

}.get(status, 'INFO')

project\_name = self.project.name if self.project else \_("بدون پروژه")

for recipient in recipients:

Notification.objects.create(

recipient=recipient,

message=f"تخصیص {self.id} - {project\_name}: {message}",

level=level

)

def save(self, \*args, \*\*kwargs):

"""مدیریت ذخیره و اعلانات"""

self.clean()

if self.allocation\_type == 'percent' and self.budget\_period:

self.allocated\_amount = (self.budget\_period.total\_amount \* self.allocated\_amount) / Decimal('100')

if not self.pk: # فقط در ایجاد اولیه

self.remaining\_amount = self.allocated\_amount

super().save(\*args, \*\*kwargs)

status, message = self.check\_allocation\_status()

if status in ('warning', 'locked', 'completed', 'stopped'):

self.send\_notification(status, message)

""" BudgetTransaction (تراکنش بودجه):"""

class BudgetTransaction(models.Model):

"""توضیح: هر تغییر در بودجه (مثل مصرف توسط تنخواه) رو ثبت می‌کنه و remaining\_amount تخصیص رو آپدیت می‌کنه."""

TRANSACTION\_TYPES = (

('ALLOCATION', \_('تخصیص اولیه')),

('CONSUMPTION', \_('مصرف')),

('ADJUSTMENT\_INCREASE', \_('افزایش تخصیص')),

('ADJUSTMENT\_DECREASE', \_('کاهش تخصیص')),

)

allocation = models.ForeignKey(BudgetAllocation, on\_delete=models.CASCADE,

related\_name='transactions', verbose\_name=\_("تخصیص بودجه"))

transaction\_type = models.CharField(max\_length=20, choices=TRANSACTION\_TYPES,

verbose\_name=\_("نوع تراکنش"))

amount = models.DecimalField(max\_digits=25, decimal\_places=2, verbose\_name=\_("مبلغ"))

related\_tankhah = models.ForeignKey('tankhah.Tankhah', on\_delete=models.SET\_NULL, null=True, blank=True,

verbose\_name=\_("تنخواه مرتبط"))

timestamp = models.DateTimeField(auto\_now\_add=True, verbose\_name=\_("زمان"))

created\_by = models.ForeignKey('accounts.CustomUser', on\_delete=models.SET\_NULL, null=True,

verbose\_name=\_("کاربر"))

description = models.TextField(blank=True, verbose\_name=\_("توضیحات"))

transaction\_id = models.CharField(max\_length=50, unique=True, verbose\_name=\_("شناسه تراکنش"))#یک شناسه منحصربه‌فرد برای هر تراکنش:

def save(self, \*args, \*\*kwargs):

from django.core.exceptions import ValidationError

if not self.transaction\_id:

self.transaction\_id = f"TX-{self.allocation.id}-{timezone.now().timestamp()}"

if self.transaction\_type == 'CONSUMPTION' and self.amount > self.allocation.remaining\_amount:

raise ValidationError(\_("مبلغ مصرف بیشتر از باقی‌مانده تخصیص است."))

if self.transaction\_type == 'RETURN':

if self.amount > self.allocation.allocated\_amount:

raise ValidationError(\_("مبلغ برگشتی نمی‌تواند بیشتر از مبلغ تخصیص‌یافته باشد."))

if not self.allocation.budget\_period.is\_active:

raise ValidationError(\_("نمی‌توان از دوره غیرفعال بودجه برگشت داد."))

self.allocation.remaining\_amount += self.amount

self.allocation.budget\_period.total\_allocated -= self.amount

self.allocation.budget\_period.save()

elif self.transaction\_type in ['CONSUMPTION', 'ADJUSTMENT\_DECREASE']:

self.allocation.remaining\_amount -= self.amount

elif self.transaction\_type in ['ALLOCATION', 'ADJUSTMENT\_INCREASE']:

self.allocation.remaining\_amount += self.amount

self.allocation.save()

super().save(\*args, \*\*kwargs)

def \_\_str\_\_(self):

return f"{self.transaction\_type} - {self.amount:,} ({self.timestamp})"

class Meta:

verbose\_name = \_("تراکنش بودجه")

verbose\_name\_plural = \_("تراکنش‌های بودجه")

default\_permissions = ()

permissions = [

('BudgetTransaction\_add', 'افزودن تراکنش بودجه'),

('BudgetTransaction\_update', 'بروزرسانی تراکنش بودجه'),

('BudgetTransaction\_delete', 'حــذف تراکنش بودجه'),

('BudgetTransaction\_view', \_('نمایش تراکنش بودجه')),

('BudgetTransaction\_return', \_('برگشت تراکنش بودجه')), # جدید

# add/update/delete معمولاً توسط سیستم انجام می‌شه، نه کاربر

]

"""PaymentOrder (دستور پرداخت):"""

class PaymentOrder(models.Model):

"""توضیح: مدل جدا برای دستور پرداخت با لینک به تنخواه و فاکتورها. min\_signatures برای کنترل تعداد امضا و شماره‌گذاری پویا با تاریخ شمسی."""

STATUS\_CHOICES = (

('DRAFT', \_('پیش‌نویس')),

('PENDING\_SIGNATURES', \_('در انتظار امضا')),

('ISSUED', \_('صادر شده')),

('PAID', \_('پرداخت شده')),

('CANCELED', \_('لغو شده')),

)

tankhah = models.ForeignKey('tankhah.Tankhah', on\_delete=models.CASCADE,

related\_name='payment\_orders', verbose\_name=\_("تنخواه"))

order\_number = models.CharField(max\_length=50, unique=True, blank=True,

verbose\_name=\_("شماره دستور پرداخت"))

issue\_date = models.DateField(default=timezone.now, verbose\_name=\_("تاریخ صدور"))

amount = models.DecimalField(max\_digits=25, decimal\_places=2, verbose\_name=\_("مبلغ"))

payee = models.ForeignKey('Payee', on\_delete=models.SET\_NULL, null=True,

verbose\_name=\_("دریافت‌کننده"))

description = models.TextField(verbose\_name=\_("شرح پرداخت"))

related\_factors = models.ManyToManyField('tankhah.Factor', blank=True,

verbose\_name=\_("فاکتورهای مرتبط"))

payment\_id = models.CharField(max\_length=50, blank=True, null=True,

verbose\_name=\_("شناسه پرداخت"))

status = models.CharField(max\_length=20, choices=STATUS\_CHOICES, default='DRAFT',

verbose\_name=\_("وضعیت"))

created\_by\_post = models.ForeignKey('core.Post', on\_delete=models.SET\_NULL, null=True,

verbose\_name=\_("پست ایجادکننده"))

min\_signatures = models.IntegerField(default=1, verbose\_name=\_("حداقل تعداد امضا"))

created\_by = models.ForeignKey('accounts.CustomUser', on\_delete=models.SET\_NULL, null=True,

related\_name='payment\_orders\_created', verbose\_name=\_("ایجادکننده"))

payment\_date = models.DateField(null=True, blank=True, verbose\_name=\_("تاریخ پرداخت"))

def generate\_order\_number(self):

sep = "-"

date\_str = jdatetime.date.fromgregorian(date=self.issue\_date).strftime('%Y%m%d')

serial = PaymentOrder.objects.filter(issue\_date=self.issue\_date).count() + 1

return f"PO{sep}{date\_str}{sep}{serial:03d}"

def save(self, \*args, \*\*kwargs):

if not self.order\_number:

self.order\_number = self.generate\_order\_number()

super().save(\*args, \*\*kwargs)

def \_\_str\_\_(self):

return f"{self.order\_number} - {self.amount}"

class Meta:

verbose\_name = \_("دستور پرداخت")

verbose\_name\_plural = \_("دستورهای پرداخت")

default\_permissions = ()

permissions = [

('PaymentOrder\_add', \_('افزودن دستور پرداخت')),

('PaymentOrder\_view', \_('نمایش دستور پرداخت')),

('PaymentOrder\_update', \_('بروزرسانی دستور پرداخت')),

('PaymentOrder\_delete', \_('حذف دستور پرداخت')),

('PaymentOrder\_sign', \_('امضای دستور پرداخت')),

('PaymentOrder\_issue', \_('صدور دستور پرداخت')),

]

"""Payee (دریافت‌کننده):"""

class Payee(models.Model):

"""توضیح: اطلاعات دریافت‌کننده پرداخت با نوع مشخص (فروشنده، کارمند، دیگر)."""

PAYEE\_TYPES = (

('VENDOR', \_('فروشنده')),

('EMPLOYEE', \_('کارمند')),

('OTHER', \_('دیگر')),

)

name = models.CharField(max\_length=100, verbose\_name=\_("نام"))

family = models.CharField(max\_length=100, verbose\_name=\_("نام خانوادگی"))

payee\_type = models.CharField(max\_length=20, choices=PAYEE\_TYPES, verbose\_name=\_("نوع"))

national\_id = models.CharField(max\_length=20, blank=True, null=True,

verbose\_name=\_("کد ملی/اقتصادی"))

account\_number = models.CharField(max\_length=50, blank=True, null=True,

verbose\_name=\_("شماره حساب"))

iban = models.CharField(max\_length=34, blank=True, null=True, verbose\_name=\_("شبا"))

address = models.TextField(blank=True, null=True, verbose\_name=\_("آدرس"))

phone = models.CharField(max\_length=20, blank=True, null=True, verbose\_name=\_("تلفن"))

created\_by = models.ForeignKey('accounts.CustomUser', on\_delete=models.SET\_NULL, null=True,

related\_name='payees\_created', verbose\_name=\_("ایجادکننده"))

is\_active= models.BooleanField(verbose\_name=\_('فعال'))

def \_\_str\_\_(self):

return f"{self.name} ({self.payee\_type})"

class Meta:

verbose\_name = \_("دریافت‌کننده")

verbose\_name\_plural = \_("دریافت‌کنندگان")

default\_permissions = ()

permissions = [

('Payee\_add', \_('افزودن دریافت‌کننده')),

('Payee\_view', \_('نمایش دریافت‌کننده')),

('Payee\_update', \_('بروزرسانی دریافت‌کننده')),

('Payee\_delete', \_('حذف دریافت‌کننده')),

]

"""TransactionType (نوع تراکنش):"""

class TransactionType(models.Model):

"""توضیح: تعریف پویا نوع تراکنش‌ها (مثل بیمه، جریمه) با امکان نیاز به تأیید اضافی."""

name = models.CharField(max\_length=250, unique=True, verbose\_name=\_("نام"))

description = models.TextField(blank=True, verbose\_name=\_("توضیحات"))

requires\_extra\_approval = models.BooleanField(default=False,

verbose\_name=\_("نیاز به تأیید اضافی"))

created\_by = models.ForeignKey('accounts.CustomUser', on\_delete=models.SET\_NULL, null=True,

related\_name='transaction\_types\_created', verbose\_name=\_("ایجادکننده"))

category = models.CharField(max\_length=50, blank=True, null=True,verbose\_name=\_('گروه بندی تراکنش‌ها '))

def \_\_str\_\_(self):

return self.name

class Meta:

verbose\_name = \_("نوع تراکنش")

verbose\_name\_plural = \_("انواع تراکنش")

default\_permissions = ()

permissions = [

('TransactionType\_add', \_('افزودن نوع تراکنش')),

('TransactionType\_view', \_('نمایش نوع تراکنش')),

('TransactionType\_update', \_('بروزرسانی نوع تراکنش')),

('TransactionType\_delete', \_('حذف نوع تراکنش')),

]

"""تخصیص بودجه به پروژه و زیر پروژه """

class ProjectBudgetAllocation(models.Model):

"""

تخصیص بودجه از BudgetAllocation شعبه به پروژه یا زیرپروژه.

توضیح: این مدل بودجه شعبه رو به پروژه یا زیرپروژه وصل می‌کنه و تاریخچه تخصیص رو نگه می‌داره.

"""

budget\_allocation = models.ForeignKey(

'budgets.BudgetAllocation', on\_delete=models.CASCADE, related\_name='project\_allocations',

verbose\_name=\_("تخصیص بودجه شعبه"))

project = models.ForeignKey('core.Project', on\_delete=models.CASCADE, related\_name='budget\_allocations',

verbose\_name=\_("پروژه"))

subproject = models.ForeignKey('core.SubProject', on\_delete=models.CASCADE, null=True, blank=True,

related\_name='budget\_allocations', verbose\_name=\_("زیرپروژه"))

allocated\_amount = models.DecimalField(max\_digits=25, decimal\_places=2, verbose\_name=\_("مبلغ تخصیص"))

allocation\_date = models.DateField(default=timezone.now, verbose\_name=\_("تاریخ تخصیص"))

created\_by = models.ForeignKey('accounts.CustomUser', on\_delete=models.SET\_NULL, null=True,

related\_name='project\_budget\_allocations\_created', verbose\_name=\_("ایجادکننده"))

description = models.TextField(blank=True, verbose\_name=\_("توضیحات"))

def get\_remaining\_amount(self):

from budgets.budget\_calculations import get\_subproject\_remaining\_budget

if self.subproject:

return get\_subproject\_remaining\_budget(self.subproject)

from budgets.budget\_calculations import get\_project\_remaining\_budget

return get\_project\_remaining\_budget(self.project)

def clean(self):

remaining = self.budget\_allocation.get\_remaining\_amount()

if self.allocated\_amount > remaining:

raise ValidationError(\_("مبلغ تخصیص بیشتر از بودجه باقی‌مانده شعبه است"))

if self.subproject and self.subproject.project != self.project:

raise ValidationError(\_("زیرپروژه باید به پروژه انتخاب‌شده تعلق داشته باشد"))

if self.allocated\_amount < 0:

raise ValidationError(\_("مبلغ تخصیص نمی‌تواند منفی باشد"))

if self.budget\_allocation and self.allocation\_date:

start\_date = self.budget\_allocation.budget\_period.start\_date

end\_date = self.budget\_allocation.budget\_period.end\_date

if not (start\_date <= self.allocation\_date <= end\_date):

raise ValidationError(\_("تاریخ تخصیص باید در بازه دوره بودجه باشد"))

def save(self, \*args, \*\*kwargs):

self.clean()

if not self.pk:

self.remaining\_amount = self.allocated\_amount

super().save(\*args, \*\*kwargs)

self.budget\_allocation.remaining\_amount = self.budget\_allocation.get\_remaining\_amount()

self.budget\_allocation.save(update\_fields=['remaining\_amount'])

self.remaining\_amount = self.get\_remaining\_amount()

super().save(update\_fields=['remaining\_amount'])

def \_\_str\_\_(self):

target = self.subproject.name if self.subproject else self.project.name

return f"{target} - {self.allocated\_amount:,} ({self.allocation\_date})"

class Meta:

verbose\_name = \_("تخصیص بودجه پروژه")

verbose\_name\_plural = \_("تخصیص‌های بودجه پروژه")

default\_permissions = ()

permissions = [

('ProjectBudgetAllocation\_add', \_('افزودن تخصیص بودجه پروژه')),

('ProjectBudgetAllocation\_view', \_('نمایش تخصیص بودجه پروژه')),

('ProjectBudgetAllocation\_update', \_('بروزرسانی تخصیص بودجه پروژه')),

('ProjectBudgetAllocation\_delete', \_('حذف تخصیص بودجه پروژه')),

('ProjectBudgetAllocation\_Head\_Office', 'تخصیص بودجه مجموعه پروژه(دفتر مرکزی)🏠'),

('ProjectBudgetAllocation\_Branch', 'تخصیص بودجه مجموعه پروژه(شعبه)🏠'),

]

"""مدل BudgetReallocation برای انتقال باقی‌مانده بودجه متوقف‌شده."""

class BudgetReallocation(models.Model):

source\_allocation = models.ForeignKey('BudgetAllocation', on\_delete=models.CASCADE, related\_name='reallocations\_from')

target\_allocation = models.ForeignKey('BudgetAllocation', on\_delete=models.CASCADE, null=True, related\_name='reallocations\_to')

amount = models.DecimalField(max\_digits=25, decimal\_places=2)

reallocation\_date = models.DateField(default=timezone.now)

reason = models.TextField()

created\_by = models.ForeignKey(CustomUser, on\_delete=models.SET\_NULL, null=True)

"""یه جدول تنظیمات (BudgetSettings) برای مدیریت قفل و هشدار در سطوح مختلف:"""

class BudgetSettings(models.Model):

level = models.CharField(max\_length=50, choices=[('PERIOD', 'دوره بودجه'), ('ALLOCATION', 'تخصیص'), ('PROJECT', 'پروژه')])

locked\_percentage = models.DecimalField(max\_digits=5, decimal\_places=2)

warning\_threshold = models.DecimalField(max\_digits=5, decimal\_places=2)

warning\_action = models.CharField(max\_length=50, choices=[('NOTIFY', 'اعلان'), ('LOCK', 'قفل'), ('RESTRICT', 'محدود')])

organization = models.ForeignKey('core.Organization', on\_delete=models.CASCADE, null=True)

budget\_period = models.ForeignKey('BudgetPeriod', on\_delete=models.CASCADE, null=True, verbose\_name=\_("دوره بودجه"))

# ردیابی مدل

# content\_type = models.ForeignKey(ContentType, on\_delete=models.CASCADE)

# object\_id = models.PositiveIntegerField()

# content\_object = GenericForeignKey('content\_type', 'object\_id')

"""مدل BudgetHistory برای لاگ کردن تغییرات بودجه و تخصیص‌ها:"""

# ------------------------------------

"""تاریخچه برای هر بودجه کلان"""

class BudgetHistory(models.Model):

content\_type = models.ForeignKey(ContentType, on\_delete=models.CASCADE)

object\_id = models.PositiveIntegerField()

# content\_object = GenericForeignKey('content\_type', 'object\_id')

action = models.CharField(max\_length=50, choices=[('CREATE', 'ایجاد'), ('UPDATE', 'بروزرسانی'), ('STOP', 'توقف'), ('REALLOCATE', 'انتقال')])

amount = models.DecimalField(max\_digits=25, decimal\_places=2, null=True)

created\_by = models.ForeignKey(CustomUser, on\_delete=models.SET\_NULL, null=True)

created\_at = models.DateTimeField(auto\_now\_add=True)

details = models.TextField(blank=True)

transaction\_type = models.CharField(max\_length=20, choices=[('ALLOCATION', 'تخصیص'), ('CONSUMPTION', 'مصرف')],

verbose\_name=\_("نوع تراکنش"))

transaction\_id = models.CharField(max\_length=50, unique=True, verbose\_name=\_("شناسه تراکنش"))

""" مدل پیشنهادی برای هزینه‌های متعارف"""

class CostCenter(models.Model):

""" مدل پیشنهادی برای هزینه‌های متعارف"""

name = models.CharField(max\_length=200, verbose\_name=\_("نام مرکز هزینه"))

code = models.CharField(max\_length=50, unique=True, verbose\_name=\_("کد مرکز هزینه"))

organization = models.ForeignKey('core.Organization', on\_delete=models.CASCADE, verbose\_name=\_("سازمان"))

budget\_allocation = models.ForeignKey(BudgetAllocation, on\_delete=models.CASCADE, verbose\_name=\_("تخصیص بودجه"))

allocated\_budget = models.DecimalField(max\_digits=25, decimal\_places=2, verbose\_name=\_("بودجه تخصیص‌یافته"))

is\_active = models.BooleanField(default=True, verbose\_name=\_("فعال"))

def get\_remaining\_budget(self):

from tankhah.models import Tankhah

consumed = Tankhah.objects.filter(cost\_center=self).aggregate(Sum('amount'))['amount\_\_sum'] or Decimal('0')

return max(self.allocated\_budget - consumed, Decimal('0'))

class Meta:

verbose\_name = \_("مرکز هزینه")

verbose\_name\_plural = \_("مراکز هزینه")

class SystemSettings(models.Model):

"""تنظیمات سیستم بودجه"""

budget\_locked\_percentage\_default = models.DecimalField(

max\_digits=5, decimal\_places=2, default=0, verbose\_name=\_("درصد قفل‌شده پیش‌فرض بودجه")

)

budget\_warning\_threshold\_default = models.DecimalField(

max\_digits=5, decimal\_places=2, default=10, verbose\_name=\_("آستانه هشدار پیش‌فرض بودجه")

)

budget\_warning\_action\_default = models.CharField(

max\_length=50, choices=[('NOTIFY', 'اعلان'), ('LOCK', 'قفل'), ('RESTRICT', 'محدود')],

default='NOTIFY', verbose\_name=\_("اقدام هشدار پیش‌فرض بودجه")

)

allocation\_locked\_percentage\_default = models.DecimalField(

max\_digits=5, decimal\_places=2, default=0, verbose\_name=\_("درصد قفل‌شده پیش‌فرض تخصیص")

)

class Meta:

verbose\_name = \_("تنظیمات سیستم")

verbose\_name\_plural = \_("تنظیمات سیستم")

def \_\_str\_\_(self):

return "تنظیمات سیستم بودجه"

مدل core

class OrganizationType(models.Model):

fname = models.CharField(max\_length=100, unique=True, null=True, blank=True, verbose\_name=\_('نام شعبه/مجتمع/اداره'))

org\_type = models.CharField(max\_length=100, unique=True, null=True, blank=True, verbose\_name=\_('نام شعبه/مجتمع/اداره'))

is\_budget\_allocatable = models.BooleanField(default=False, verbose\_name=\_("قابل استفاده برای تخصیص بودجه"))

def \_\_str\_\_(self):

return f"{self.fname} - {self.org\_type} "or \_("نامشخص")

class Meta:

verbose\_name = \_('عنوان مرکز/شعبه/اداره/سازمان')

verbose\_name\_plural = \_('عنوان مرکز/شعبه/اداره/سازمان')

default\_permissions = ()

permissions = [

('OrganizationType\_add', 'افزودن شعبه/اداره/مجتمع/سازمان'),

('OrganizationType\_view', 'نمایش شعبه/اداره/مجتمع/سازمان'),

('OrganizationType\_update', 'ویرایش شعبه/اداره/مجتمع/سازمان'),

('OrganizationType\_delete', 'حــذف شعبه/اداره/مجتمع/سازمان'),

]

class Organization(models.Model):

"""مدل سازمان برای تعریف مجتمع‌ها و دفتر مرکزی"""

code = models.CharField(max\_length=10, unique=True, verbose\_name=\_("کد سازمان"))

name = models.CharField(max\_length=100, verbose\_name=\_("نام سازمان"))

org\_type = models.ForeignKey(

'OrganizationType',

on\_delete=models.SET\_NULL,

null=True,

blank=True,

verbose\_name=\_("نوع سازمان"),

related\_name='organizations' # اضافه کردن related\_name برای وضوح

)

description = models.TextField(blank=True, null=True, verbose\_name=\_("توضیحات"))

parent\_organization = models.ForeignKey(

'self',

on\_delete=models.SET\_NULL,

null=True,

blank=True,

verbose\_name=\_("سازمان والد")

)

is\_active = models.BooleanField(default=True, verbose\_name=\_("فعال"))

is\_core = models.BooleanField(default=False, verbose\_name=\_("دفتر مرکزی سازمان")) # تغییر پیش‌فرض به False

def clean(self):

"""اعتبارسنجی مدل برای اطمینان از منطق دفتر مرکزی"""

from django.core.exceptions import ValidationError

if self.is\_core and self.parent\_organization:

raise ValidationError(\_('دفتر مرکزی نمی‌تواند سازمان والد داشته باشد.'))

if self.is\_core:

# بررسی وجود تنها یک دفتر مرکزی فعال

existing\_core = Organization.objects.filter(is\_core=True, is\_active=True).exclude(pk=self.pk)

if existing\_core.exists():

raise ValidationError(\_('فقط یک سازمان می‌تواند به‌عنوان دفتر مرکزی فعال باشد.'))

def \_\_str\_\_(self):

org\_type\_str = self.org\_type.fname if self.org\_type else \_("نامشخص")

return f"{self.code} - {self.name} ({org\_type\_str})"

@property

def org\_type\_code(self):

return self.org\_type.fname if self.org\_type else None

def save(self, \*args, \*\*kwargs):

"""اجرای clean قبل از ذخیره"""

self.full\_clean() # اجرای اعتبارسنجی

super().save(\*args, \*\*kwargs)

class Meta:

verbose\_name = \_("سازمان")

verbose\_name\_plural = \_("سازمان‌ها")

default\_permissions = ()

permissions = [

('Organization\_add', 'افزودن سازمان برای تعریف مجتمع‌ها و دفتر مرکزی'),

('Organization\_update', 'بروزرسانی سازمان برای تعریف مجتمع‌ها و دفتر مرکزی'),

('Organization\_delete', 'حــذف سازمان برای تعریف مجتمع‌ها و دفتر مرکزی'),

('Organization\_view', 'نمایش سازمان برای تعریف مجتمع‌ها و دفتر مرکزی'),

]

indexes = [

models.Index(fields=['code', 'org\_type']),

]

class Project(models.Model):

name = models.CharField(max\_length=100, verbose\_name=\_("نام پروژه"))

code = models.CharField(max\_length=80, unique=True, verbose\_name=\_("کد پروژه"))

organizations = models.ManyToManyField(Organization, limit\_choices\_to={'org\_type': 'COMPLEX'}, verbose\_name=\_("مجتمع‌های مرتبط"))

# allocations = models.ManyToManyField('budgets.BudgetAllocation', blank=True, verbose\_name=\_("تخصیص‌های بودجه مرتبط"))

start\_date = models.DateField(verbose\_name=\_("تاریخ شروع"))

end\_date = models.DateField(null=True, blank=True, verbose\_name=\_("تاریخ پایان"))

description = models.TextField(blank=True, null=True, verbose\_name=\_("توضیحات"))

is\_active = models.BooleanField(default=True, verbose\_name="وضعیت فعال")

PRIORITY\_CHOICES = (('LOW', \_('کم')), ('MEDIUM', \_('متوسط')), ('HIGH', \_('زیاد')),)

priority = models.CharField(max\_length=20, choices=PRIORITY\_CHOICES, default='MEDIUM', verbose\_name=\_("اولویت"))

def get\_total\_budget(self):

return get\_project\_total\_budget(self)

def get\_remaining\_budget(self):

return get\_project\_remaining\_budget(self)

def \_\_str\_\_(self):

status = "فعال" if self.is\_active else "غیرفعال"

return f"{self.code} - {self.name} ({status})"

class Meta:

verbose\_name = \_("پروژه")

verbose\_name\_plural = \_("پروژه")

default\_permissions = ()

permissions = [

('Project\_add', 'افزودن مجموعه پروژه'),

('Project\_update', 'ویرایش مجموعه پروژه'),

('Project\_view', 'نمایش مجموعه پروژه'),

('Project\_delete', 'حــذف مجموعه پروژه'),

# ('Project\_Budget\_allocation\_Head\_Office', 'تخصیص بودجه مجموعه پروژه(دفتر مرکزی)'),

# ('Project\_Budget\_allocation\_Branch', 'تخصیص بودجه مجموعه پروژه(شعبه)'),

]

class SubProject(models.Model):

project = models.ForeignKey(Project, on\_delete=models.CASCADE, related\_name='subprojects', verbose\_name=\_("پروژه اصلی"))

name = models.CharField(max\_length=200, verbose\_name=\_("نام ساب‌پروژه"))

description = models.TextField(blank=True, null=True, verbose\_name=\_("توضیحات"))

allocated\_budget = models.DecimalField(max\_digits=25, decimal\_places=2, default=0,

verbose\_name=\_("بودجه تخصیص‌یافته"))

allocations = models.ManyToManyField('budgets.ProjectBudgetAllocation',

related\_name='budget\_allocations\_set' ,blank=True, verbose\_name=\_("تخصیص‌های بودجه مرتبط"))

is\_active = models.BooleanField(default=True, verbose\_name=\_("فعال"))

def get\_remaining\_budget(self):

return get\_subproject\_remaining\_budget(self)

def save(self, \*args, \*\*kwargs):

super().save(\*args, \*\*kwargs)

if not self.pk:

total\_allocated = sum([alloc.amount for alloc in self.allocations.all()])

if total\_allocated > self.project.get\_remaining\_budget():

raise ValueError("بودجه تخصیص‌یافته بیشتر از بودجه باقی‌مانده پروژه است.")

def \_\_str\_\_(self):

return f"{self.name} ({self.project.name})"

class Meta:

verbose\_name = \_("ساب‌پروژه")

verbose\_name\_plural = \_("ساب‌پروژه‌ها")

default\_permissions =()

permissions = [

('SubProject\_add','افزودن زیر مجموعه پروژه'),

('SubProject\_update','ویرایش زیر مجموعه پروژه'),

('SubProject\_view','نمایش زیر مجموعه پروژه'),

('SubProject\_delete','حــذف زیر مجموعه پروژه'),

('SubProject\_Head\_Office','تخصیص زیر مجموعه پروژه(دفتر مرکزی)🏠'),

('SubProject\_Branch','تخصیص زیر مجموعه پروژه(شعبه)🏠'),

]

مدل tankhah

class Tankhah(models.Model):

"""مدل تنخواه برای ثبت و مدیریت درخواست‌های مالی"""

STATUS\_CHOICES = (

('DRAFT', \_('پیش‌نویس')),

('PENDING', \_('در حال بررسی')),

('APPROVED', \_('تأییدشده')),

('SENT\_TO\_HQ', \_('ارسال‌شده به HQ')),

('HQ\_OPS\_PENDING', \_('در حال بررسی - بهره‌برداری')),

('HQ\_OPS\_APPROVED', \_('تأییدشده - بهره‌برداری')),

('HQ\_FIN\_PENDING', \_('در حال بررسی - مالی')),

('PAID', \_('پرداخت‌شده')),

('REJECTED', \_('ردشده')),

)

number = models.CharField(max\_length=50, unique=True, blank=True, verbose\_name=\_("شماره تنخواه"))

amount = models.DecimalField(max\_digits=25, decimal\_places=2, verbose\_name=\_("مبلغ"))

date = models.DateTimeField(default=timezone.now, verbose\_name=\_("تاریخ"))

due\_date = models.DateTimeField(null=True, blank=True, verbose\_name=\_('مهلت زمانی'))

created\_at = models.DateTimeField(auto\_now\_add=True, verbose\_name=\_("تاریخ ایجاد")) # اجبار در توقف

organization = models.ForeignKey('core.Organization', on\_delete=models.CASCADE, verbose\_name=\_('مجموعه/شعبه'))

# project = models.ForeignKey('core.Project', on\_delete=models.SET\_NULL, null=True, blank=True, verbose\_name=\_('پروژه'))

project = models.ForeignKey('core.Project', on\_delete=models.SET\_NULL, null=True, blank=True, related\_name='tankhah\_set', verbose\_name=\_('پروژه'))

subproject = models.ForeignKey(SubProject, on\_delete=models.CASCADE, null=True, blank=True, verbose\_name=\_("زیر مجموعه پروژه"))

letter\_number = models.CharField(max\_length=50, blank=True, null=True, verbose\_name=\_("شماره نامه"))

created\_by = models.ForeignKey(CustomUser, on\_delete=models.SET\_NULL, null=True, related\_name='tankhah\_created', verbose\_name=\_("ایجادکننده"))

approved\_by = models.ManyToManyField(CustomUser, blank=True, verbose\_name=\_('تأییدکنندگان'))

description = models.TextField(verbose\_name=\_("توضیحات"))

current\_stage = models.ForeignKey(WorkflowStage, on\_delete=models.SET\_NULL, null=True,

default=get\_default\_workflow\_stage, verbose\_name="مرحله فعلی")

status = models.CharField(max\_length=20, choices=STATUS\_CHOICES, default='DRAFT', verbose\_name=\_("وضعیت"))

hq\_status = models.CharField(max\_length=20, default='PENDING',

choices=STATUS\_CHOICES, null=True, blank=True,

verbose\_name=\_("وضعیت در HQ"))

last\_stopped\_post = models.ForeignKey(Post, null=True, blank=True, on\_delete=models.SET\_NULL, verbose\_name=\_("آخرین پست متوقف‌شده"))

is\_archived = models.BooleanField(default=False, verbose\_name=\_("آرشیو شده"))

payment\_number = models.CharField(max\_length=50, blank=True, null=True, verbose\_name=\_("شماره پرداخت"))

is\_locked = models.BooleanField(default=False, verbose\_name=\_("قفل شده"))

archived\_at = models.DateTimeField(null=True, blank=True, verbose\_name="زمان آرشیو")

canceled = models.BooleanField(default=False, verbose\_name="لغو شده")

remaining\_budget = models.DecimalField(max\_digits=25, decimal\_places=2, default=0, verbose\_name=\_("بودجه باقیمانده"))

budget\_allocation = models.ForeignKey(BudgetAllocation, on\_delete=models.SET\_NULL, null=True,

verbose\_name=\_("تخصیص بودجه"))

is\_emergency = models.BooleanField(default=False, verbose\_name=\_("اضطراری"))

"""توضیح: لینک به BudgetAllocation برای مصرف بودجه و اضافه شدن is\_emergency."""

request\_date = models.DateField(default=timezone.now, verbose\_name=\_("تاریخ درخواست"))

def generate\_number(self):

"""تولید شماره یکتا برای تنخواه با تاریخ شمسی"""

sep = NUMBER\_SEPARATOR

# تبدیل تاریخ میلادی به شمسی

import jdatetime

jalali\_date = jdatetime.datetime.fromgregorian(datetime=self.date)

date\_str = jalali\_date.strftime('%Y%m%d') # فرمت YYYYMMDD شمسی

org\_code = self.organization.code

project\_code = self.project.code if self.project else 'NOPRJ'

# پیدا کردن بالاترین شماره سریال برای این تاریخ و سازمان

with transaction.atomic():

max\_serial = Tankhah.objects.filter(

organization=self.organization,

date\_\_date=self.date.date() # همچنان از تاریخ میلادی برای فیلتر استفاده می‌کنیم

).aggregate(Max('number'))['number\_\_max']

if max\_serial:

# استخراج شماره سریال از آخرین شماره موجود

last\_number = max\_serial.split(sep)[-1]

serial = int(last\_number) + 1

else:

serial = 1

new\_number = f"TNKH{sep}{date\_str}{sep}{org\_code}{sep}{project\_code}{sep}{serial:03d}"

# چک کردن یکتایی و افزایش سریال در صورت نیاز

while Tankhah.objects.filter(number=new\_number).exists():

serial += 1

new\_number = f"TNKH{sep}{date\_str}{sep}{org\_code}{sep}{project\_code}{sep}{serial:03d}"

return new\_number

def save(self, \*args, \*\*kwargs):

if not self.number:

self.number = self.generate\_number()

# اگه وضعیت COMPLETED یا PAID باشه، قفل کن

if self.status in ['COMPLETED', 'PAID'] and not self.is\_locked:

self.is\_locked = True

from budgets.models import ProjectBudgetAllocation

if not self.pk: # فقط موقع ایجاد

# پیدا کردن تخصیص بودجه پروژه/زیرپروژه

if self.subproject:

allocation = ProjectBudgetAllocation.objects.filter(

budget\_allocation=self.budget\_allocation,

subproject=self.subproject

).first()

budget = allocation.remaining\_amount if allocation else self.subproject.get\_remaining\_budget()

elif self.project:

allocation = ProjectBudgetAllocation.objects.filter(

budget\_allocation=self.budget\_allocation,

project=self.project,

subproject\_\_isnull=True

).first()

budget = allocation.remaining\_amount if allocation else self.project.get\_remaining\_budget()

else:

raise ValueError("تنخواه باید به پروژه یا ساب‌پروژه وصل باشد.")

if self.amount > budget:

raise ValueError(f"مبلغ تنخواه ({self.amount}) بیشتر از بودجه باقیمانده ({budget}) است.")

# به‌روزرسانی بودجه تخصیص پروژه

if allocation:

allocation.remaining\_amount -= self.amount

allocation.save()

# ثبت تراکنش بودجه

BudgetTransaction.objects.create(

allocation=self.budget\_allocation,

transaction\_type='CONSUMPTION',

amount=self.amount,

related\_tankhah=self,

created\_by=self.created\_by,

description=f"مصرف بودجه توسط تنخواه {self.number}"

)

super().save(\*args, \*\*kwargs)

def \_\_str\_\_(self):

project\_str = self.project.name if self.project else 'بدون پروژه'

subproject\_str = f" ({self.subproject.name})" if self.subproject else ''

return f"{self.number} - {project\_str}{subproject\_str}"

class Meta:

verbose\_name = \_("تنخواه")

verbose\_name\_plural = \_("تنخواه‌ها")

indexes = [

models.Index(fields=['number', 'date', 'status' ,'organization'])]

default\_permissions =()

permissions = [

('Tankhah\_add', 'ثبت تنخواه'),

('Tankhah\_update', 'بروزرسانی تنخواه'),

('Tankhah\_view', 'نمایش تنخواه'),

('Tankhah\_delete', 'حذف تنخواه'),

('Tankhah\_part\_approve', '👍تأیید رئیس قسمت'),

('Tankhah\_approve', '👍تأیید مدیر مجموعه'),

('Tankhah\_hq\_view', 'رصد دفتر مرکزی'),

('Tankhah\_hq\_approve', '👍تأیید رده بالا در دفتر مرکزی'),

('Tankhah\_HQ\_OPS\_PENDING', \_('در حال بررسی - بهره‌برداری')),

('Tankhah\_HQ\_OPS\_APPROVED', \_('👍تأییدشده - بهره‌برداری')),

('Tankhah\_HQ\_FIN\_PENDING', \_('در حال بررسی - مالی')),

('Tankhah\_PAID', \_('پرداخت‌شده')),

('Tankhah\_REJECTED', \_('ردشده')),

("FactorItem\_approve", "👍تایید/رد ردیف فاکتور (تایید ردیف فاکتور\*استفاده در مراحل تایید\*)"),

('edit\_full\_tankhah','👍😊تغییرات کاربری در فاکتور /تایید یا رد ردیف ها '),

('Dashboard\_Core\_view', 'دسترسی به داشبورد Core پایه'),

('DashboardView\_flows\_view', 'دسترسی به روند تنخواه گردانی'),

('Dashboard\_\_view', 'دسترسی به داشبورد اصلی 💻'),

('Dashboard\_Stats\_view', 'دسترسی به آمار کلی داشبورد💲'),

]

class Factor(models.Model):

"""مدل فاکتور برای جزئیات تنخواه"""

STATUS\_CHOICES = (

('PENDING', \_('در حال بررسی')),

('APPROVED', \_('تأییدشده')),

('REJECTED', \_('ردشده')),

)

number = models.CharField(max\_length=60, blank=True, verbose\_name=\_("شماره فاکتور"))

tankhah = models.ForeignKey(Tankhah, on\_delete=models.PROTECT, related\_name='factors', verbose\_name=\_("تنخواه"))

date = models.DateField(default=timezone.now, verbose\_name=\_("تاریخ"))

amount = models.DecimalField(max\_digits=20, decimal\_places=2, verbose\_name=\_('مبلغ فاکتور'), default=0) # فرض بر وجود فیلد مبلغ

description = models.TextField(verbose\_name=\_("توضیحات"))

status = models.CharField(max\_length=20, choices=STATUS\_CHOICES, default='PENDING', verbose\_name=\_("وضعیت"))

approved\_by = models.ManyToManyField(CustomUser, blank=True, verbose\_name=\_("تأییدکنندگان"))

is\_finalized = models.BooleanField(default=False, verbose\_name=\_("نهایی شده"))

locked = models.BooleanField(default=False, verbose\_name="قفل شده")

locked\_by\_stage = models.ForeignKey(WorkflowStage, null=True, blank=True, on\_delete=models.SET\_NULL, verbose\_name=\_("قفل شده توسط مرحله"))

budget = models.DecimalField( max\_digits=20, decimal\_places=2, default=0, verbose\_name=\_("بودجه تخصیصی"))

remaining\_budget = models.DecimalField(max\_digits=20, decimal\_places=2, default=0,

verbose\_name=\_("بودجه باقیمانده"))

def generate\_number(self):

"""تولید شماره فاکتور با جداکننده قابل تنظیم"""

sep = NUMBER\_SEPARATOR

serial = self.tankhah.factors.count() + 1

return f"{self.tankhah.number}{sep}F{serial}"

def save(self, \*args, \*\*kwargs):

if not self.number:

sep = "-"

serial = self.tankhah.factors.count() + 1

# self.number = f"{self.tanbakh.number}{sep}F{serial}"

self.number = self.generate\_number()

if not self.pk:

self.remaining\_budget = self.budget

if self.total\_amount() > self.budget:

raise ValueError("مبلغ فاکتور نمی‌تواند بیشتر از بودجه تخصیصی باشد")

super().save(\*args, \*\*kwargs)

def total\_amount(self):

sum\_Factor=sum(item.amount for item in self.items.all()) if self.items.exists() else 0

logging.info(f'sum\_factor is {sum\_Factor}')

return sum\_Factor

def \_\_str\_\_(self):

return f"{self.number} ({self.tankhah.number})"

class Meta:

verbose\_name = \_("فاکتور")

verbose\_name\_plural = \_("فاکتورها")

indexes = [

models.Index(fields=['number', 'date', 'status']),

]

default\_permissions=()

permissions = [

('a\_factor\_add','افزودن فاکتور برای جزئیات تنخواه '),

('a\_factor\_update','ویرایش فاکتور برای جزئیات تنخواه'),

('a\_factor\_delete','حــذف فاکتور برای جزئیات تنخواه'),

('a\_factor\_view','نمایش فاکتور برای جزئیات تنخواه'),

]

class FactorItem(models.Model):

""" اقلام فاکتور """

STATUS\_CHOICES = (

('PENDING', \_('در حال بررسی')),

('APPROVED', \_('تأیید شده')),

('REJECTED', \_('رد شده')),

('PAID', 'پرداخت شده'),

)

factor = models.ForeignKey(Factor, on\_delete=models.CASCADE, related\_name='items', verbose\_name=\_("فاکتور"))

description = models.CharField(max\_length=255, verbose\_name=\_("شرح ردیف"))

amount = models.DecimalField(max\_digits=25, decimal\_places=2, verbose\_name=\_("مبلغ") )

status = models.CharField(max\_length=20, choices=STATUS\_CHOICES, default='PENDING', verbose\_name=\_("وضعیت"))

quantity = models.DecimalField(max\_digits=25, default=1,decimal\_places=2, verbose\_name=\_("تعداد"))

# quantity = models.IntegerField(default=1, verbose\_name=\_("تعداد"))

unit\_price = models.DecimalField(max\_digits=25,default=1, decimal\_places=1, verbose\_name=\_("قیمت واحد"))

category = models.CharField(max\_length=100, blank=True, null=True, verbose\_name=\_("دسته‌بندی"))

transaction\_type = models.ForeignKey(TransactionType, on\_delete=models.SET\_NULL, null=True,

verbose\_name=\_("نوع تراکنش"))

min\_stage\_order = models.IntegerField(default=1, verbose\_name=\_("حداقل ترتیب مرحله"),

help\_text=\_("این نوع تراکنش فقط در این مرحله یا بالاتر مجاز است"))

# حذف TRANSACTION\_TYPES و استفاده از مدل TransactionType

def save(self, \*args, \*\*kwargs):

if not self.amount: # اگه amount وارد نشده باشه، محاسبه کن

self.amount = self.unit\_price \* self.quantity

# self.amount = self.unit\_price \* self.quantity

# چک کن که مرحله فعلی تنخواه برای این نوع تراکنش مجاز باشه

if self.factor.tankhah.current\_stage.order < self.min\_stage\_order:

raise ValueError(

f"تراکنش نوع {self.transaction\_type} فقط در مرحله {self.min\_stage\_order} یا بالاتر مجاز است")

self.amount = self.unit\_price \* self.quantity

if self.amount > self.factor.remaining\_budget:

raise ValueError("مبلغ ردیف بیشتر از بودجه باقیمانده فاکتور است")

super().save(\*args, \*\*kwargs)

self.factor.remaining\_budget -= self.amount

self.factor.tankhah.remaining\_budget -= self.amount

self.factor.save()

self.factor.tankhah.save()

super().save(\*args, \*\*kwargs)

def \_\_str\_\_(self):

return f"{self.description} - {self.amount}"

class Meta:

verbose\_name = \_("ردیف فاکتور")

verbose\_name\_plural = \_("ردیف‌های فاکتور")

default\_permissions = ()

permissions = [

('FactorItem\_add','افزودن اقلام فاکتور'),

('FactorItem\_update','ویرایش اقلام فاکتور'),

('FactorItem\_view','نمایش اقلام فاکتور'),

('FactorItem\_delete','حذف اقلام فاکتور'),

]

# Create your models here.

# -- New

# ------------------------------------

"""BudgetPeriod (دوره بودجه کلان):"""

class BudgetPeriod(models.Model):

organization = models.ForeignKey('core.Organization', on\_delete=models.CASCADE, verbose\_name=\_("دفتر مرکزی"),

related\_name='budget\_periods')

name = models.CharField(max\_length=100, unique=True, verbose\_name=\_("نام دوره بودجه"))

start\_date = models.DateField(verbose\_name=\_("تاریخ شروع"))

end\_date = models.DateField(verbose\_name=\_("تاریخ پایان"))

total\_amount = models.DecimalField(max\_digits=25, decimal\_places=0, verbose\_name=\_("مبلغ کل"))

total\_allocated = models.DecimalField(max\_digits=25, decimal\_places=2, default=0, verbose\_name=\_("مجموع تخصیص‌ها"))

is\_active = models.BooleanField(default=True, verbose\_name=\_("فعال"))

is\_archived = models.BooleanField(default=False, verbose\_name=\_("بایگانی شده"))

is\_completed = models.BooleanField(default=False, verbose\_name=\_("تمام‌شده"))

created\_by = models.ForeignKey('accounts.CustomUser', on\_delete=models.SET\_NULL, null=True,

related\_name='budget\_periods\_created', verbose\_name=\_("ایجادکننده"))

locked\_percentage = models.IntegerField(default=0, verbose\_name=\_("درصد قفل‌شده"),

help\_text=\_("درصد بودجه که قفل می‌شود (0-100)"))

warning\_threshold = models.DecimalField(max\_digits=5, decimal\_places=2, default=10, verbose\_name=\_("آستانه اخطار"),

help\_text=\_("درصدی که هشدار نمایش داده می‌شود (0-100)"))

lock\_condition = models.CharField(max\_length=50,

choices=[('AFTER\_DATE', \_("بعد از تاریخ پایان")), ('MANUAL', \_("دستی")),

('ZERO\_REMAINING', \_("باقی‌مانده صفر")), ], default='AFTER\_DATE',

verbose\_name=\_("شرط قفل"))

warning\_action = models.CharField(max\_length=50, choices=[('NOTIFY', \_("فقط اعلان")), ('LOCK', \_("قفل کردن")),

('RESTRICT', \_("محدود کردن ثبت")), ], default='NOTIFY',

verbose\_name=\_("اقدام هشدار"),

help\_text=\_("رفتار سیستم هنگام رسیدن به آستانه هشدار"))

allocation\_phase = models.CharField(max\_length=50, blank=True, verbose\_name=\_("فاز تخصیص"))

description = models.TextField(blank=True, verbose\_name=\_("توضیحات"))

returned\_amount = models.DecimalField(max\_digits=25, decimal\_places=2, default=0, verbose\_name=\_("مجموع بودجه برگشتی"))

class Meta:

verbose\_name = \_("دوره بودجه")

verbose\_name\_plural = \_("دوره‌های بودجه")

default\_permissions = ()

permissions = [

('budgetperiod\_add', \_("افزودن دوره بودجه")),

('budgetperiod\_update', \_("بروزرسانی دوره بودجه")),

('budgetperiod\_view', \_("نمایش دوره بودجه")),

('budgetperiod\_delete', \_("حذف دوره بودجه")),

('budgetperiod\_archive', \_("بایگانی دوره بودجه")),

]

def \_\_str\_\_(self):

return f"{self.name} ({self.organization.code})"

def clean(self):

# super().clean()

# if self.total\_allocated > self.total\_amount:

# raise ValidationError(\_("مجموع تخصیص‌ها نمی‌تواند از مبلغ کل بیشتر باشد."))

if not self.start\_date or not self.end\_date:

raise ValidationError(\_("تاریخ شروع و پایان نمی‌توانند خالی باشند."))

if self.end\_date <= self.start\_date:

raise ValidationError(\_("تاریخ پایان باید بعد از تاریخ شروع باشد."))

if not (0 <= self.locked\_percentage <= 100):

raise ValidationError(\_("درصد قفل‌شده باید بین 0 تا 100 باشد."))

if not (0 <= self.warning\_threshold <= 100):

raise ValidationError(\_("آستانه هشدار باید بین 0 تا 100 باشد."))

if self.is\_completed and self.is\_active:

raise ValidationError(\_("دوره تمام‌شده نمی‌تواند فعال باشد."))

def get\_remaining\_amount(self):

from budgets.models import BudgetAllocation

allocated = BudgetAllocation.objects.filter(budget\_period=self).aggregate(

total=Sum('allocated\_amount')

)['total'] or Decimal('0')

return max(self.total\_amount - allocated, Decimal('0'))

def get\_locked\_amount(self):

return (self.total\_amount \* self.locked\_percentage) / Decimal('100')

def get\_warning\_amount(self):

return (self.total\_amount \* self.warning\_threshold) / Decimal('100')

def send\_notification(self, status, message):

from notifications.models import Notification

from accounts.models import CustomUser

recipients = CustomUser.objects.filter(

models.Q(organization=self.organization) |

models.Q(organization\_\_parent=self.organization)

).distinct()

level = {

'warning': 'WARNING',

'locked': 'ERROR',

'completed': 'ERROR',

'inactive': 'INFO',

'normal': 'INFO'

}.get(status, 'INFO')

for recipient in recipients:

Notification.objects.create(

recipient=recipient,

message=f"{self.name}: {message}",

level=level

)

def save(self, \*args, \*\*kwargs):

self.full\_clean()

super().save(\*args, \*\*kwargs)

status, message = check\_budget\_status(self)

if status in ('warning', 'locked', 'completed'):

self.send\_notification(status, message)

# ------------------------------------

""" BudgetAllocation (تخصیص بودجه):"""

class BudgetAllocation(models.Model):

"""

تخصیص بودجه به سازمان‌ها (شعبات یا ادارات) و پروژه‌ها.

قابلیت‌ها: تخصیص چندباره، توقف بودجه، و ردیابی هزینه‌ها.

"""

budget\_period = models.ForeignKey('BudgetPeriod', on\_delete=models.CASCADE, related\_name='allocations',

verbose\_name=\_("دوره بودجه"))

organization = models.ForeignKey('core.Organization', on\_delete=models.CASCADE, related\_name='budget\_allocations',

verbose\_name=\_("سازمان دریافت‌کننده"))

project = models.ForeignKey('core.Project', on\_delete=models.CASCADE, related\_name='allocations',

verbose\_name=\_("پروژه"), null=True, blank=True) # اختیاری کردن پروژه

allocated\_amount = models.DecimalField(max\_digits=25, decimal\_places=2, verbose\_name=\_("مبلغ تخصیص"))

remaining\_amount = models.DecimalField(max\_digits=25, decimal\_places=2, default=0,

verbose\_name=\_("باقی‌مانده تخصیص"))

allocation\_date = models.DateField(default=timezone.now, verbose\_name=\_("تاریخ تخصیص"))

created\_by = models.ForeignKey('accounts.CustomUser', on\_delete=models.SET\_NULL, null=True,

related\_name='budget\_allocations\_created', verbose\_name=\_("ایجادکننده"))

description = models.TextField(blank=True, verbose\_name=\_("توضیحات"))

is\_active = models.BooleanField(default=True, verbose\_name=\_("فعال"))

is\_stopped = models.BooleanField(default=False, verbose\_name=\_("متوقف‌شده"))

ALLOCATION\_TYPES = (('amount', \_("مبلغ ثابت")), ('percent', \_("درصد")),)

allocation\_type = models.CharField(max\_length=20, choices=ALLOCATION\_TYPES, default='amount',

verbose\_name=\_("نوع تخصیص"))

locked\_percentage = models.DecimalField(max\_digits=5, decimal\_places=2, default=0, verbose\_name=\_("درصد قفل‌شده"),

help\_text=\_("درصد تخصیص که قفل می‌شود (0-100)"))

warning\_threshold = models.DecimalField(max\_digits=5, decimal\_places=2, default=10, verbose\_name=\_("آستانه اخطار"),

help\_text=\_("درصدی که هشدار نمایش داده می‌شود (0-100)"))

warning\_action = models.CharField(

max\_length=50,

choices=[('NOTIFY', \_("فقط اعلان")), ('LOCK', \_("قفل کردن")), ('RESTRICT', \_("محدود کردن ثبت")), ],

default='NOTIFY',

verbose\_name=\_("اقدام هشدار"),

help\_text=\_("رفتار سیستم هنگام رسیدن به آستانه هشدار")

)

allocation\_number = models.IntegerField(default=1, verbose\_name=\_("شماره تخصیص"))

returned\_amount = models.DecimalField(max\_digits=25, decimal\_places=2, default=0,verbose\_name=\_("مجموع بودجه برگشتی"))

class Meta:

verbose\_name = \_("تخصیص بودجه")

verbose\_name\_plural = \_("تخصیص‌های بودجه")

default\_permissions = ()

permissions = [

('budgetallocation\_add', \_("افزودن تخصیص بودجه")),

('budgetallocation\_view', \_("نمایش تخصیص بودجه")),

('budgetallocation\_update', \_("بروزرسانی تخصیص بودجه")),

('budgetallocation\_delete', \_("حذف تخصیص بودجه")),

('budgetallocation\_adjust', \_("تنظیم تخصیص بودجه (افزایش/کاهش)")),

('budgetallocation\_stop', \_("توقف تخصیص بودجه")),

('budgetallocation\_return', \_("برگشت تخصیص بودجه")), # جدید

]

indexes = [

models.Index(fields=['budget\_period', 'allocation\_date']),

models.Index(fields=['organization', 'allocated\_amount']),

]

def \_\_str\_\_(self):

jalali\_date = jdatetime.date.fromgregorian(date=self.allocation\_date).strftime('%Y/%m/%d')

project\_name = self.project.name if self.project else \_("بدون پروژه")

return f"تخصیص {self.id} - {self.organization.name} - {project\_name} - {self.allocated\_amount:,} ({jalali\_date})"

def clean(self):

super().clean()

if self.allocated\_amount <= 0:

raise ValidationError(\_("مبلغ تخصیص باید مثبت باشد."))

if not (0 <= self.locked\_percentage <= 100):

raise ValidationError(\_("درصد قفل‌شده باید بین 0 تا 100 باشد."))

if not (0 <= self.warning\_threshold <= 100):

raise ValidationError(\_("آستانه اخطار باید بین 0 تا 100 باشد."))

if self.allocation\_type == 'percent' and self.allocated\_amount > 100:

raise ValidationError(\_("درصد تخصیص نمی‌تواند بیشتر از 100 باشد."))

if self.budget\_period and self.allocation\_date:

allocation\_date = self.allocation\_date

if hasattr(allocation\_date, "date"):

allocation\_date = allocation\_date.date()

if not (self.budget\_period.start\_date <= allocation\_date <= self.budget\_period.end\_date):

raise ValidationError(\_("تاریخ تخصیص باید در بازه دوره بودجه باشد."))

if self.organization.is\_core:

raise ValidationError(\_("نمی‌توان به دفتر مرکزی تخصیص داد."))

def get\_remaining\_amount(self):

return max(self.remaining\_amount, Decimal('0'))

def get\_locked\_amount(self):

return (self.allocated\_amount \* self.locked\_percentage) / Decimal('100')

def get\_warning\_amount(self):

return (self.allocated\_amount \* self.warning\_threshold) / Decimal('100')

def check\_allocation\_status(self):

remaining = self.get\_remaining\_amount()

locked = self.get\_locked\_amount()

warning = self.get\_warning\_amount()

if not self.is\_active:

return 'inactive', \_('تخصیص غیرفعال است.')

if self.is\_stopped:

return 'stopped', \_('تخصیص متوقف شده است.')

if remaining <= 0:

return 'completed', \_('تخصیص تمام‌شده است.')

if remaining <= locked:

return 'locked', \_('تخصیص به حد قفل‌شده رسیده است.')

if remaining <= warning:

return 'warning', \_('تخصیص به آستانه هشدار رسیده است.')

return 'normal', \_('وضعیت عادی')

def send\_notification(self, status, message):

from tankhah.models import Notification

recipients = CustomUser.objects.filter(

models.Q(organization=self.organization) |

models.Q(organization\_\_parent=self.organization)

).distinct()

level = {

'warning': 'WARNING',

'locked': 'ERROR',

'completed': 'ERROR',

'stopped': 'ERROR',

'inactive': 'INFO',

'normal': 'INFO'

}.get(status, 'INFO')

project\_name = self.project.name if self.project else \_("بدون پروژه")

for recipient in recipients:

Notification.objects.create(

recipient=recipient,

message=f"تخصیص {self.id} - {project\_name}: {message}",

level=level

)

def save(self, \*args, \*\*kwargs):

"""مدیریت ذخیره و اعلانات"""

self.clean()

if self.allocation\_type == 'percent' and self.budget\_period:

self.allocated\_amount = (self.budget\_period.total\_amount \* self.allocated\_amount) / Decimal('100')

if not self.pk: # فقط در ایجاد اولیه

self.remaining\_amount = self.allocated\_amount

super().save(\*args, \*\*kwargs)

status, message = self.check\_allocation\_status()

if status in ('warning', 'locked', 'completed', 'stopped'):

self.send\_notification(status, message)

""" BudgetTransaction (تراکنش بودجه):"""

class BudgetTransaction(models.Model):

"""توضیح: هر تغییر در بودجه (مثل مصرف توسط تنخواه) رو ثبت می‌کنه و remaining\_amount تخصیص رو آپدیت می‌کنه."""

TRANSACTION\_TYPES = (

('ALLOCATION', \_('تخصیص اولیه')),

('CONSUMPTION', \_('مصرف')),

('ADJUSTMENT\_INCREASE', \_('افزایش تخصیص')),

('ADJUSTMENT\_DECREASE', \_('کاهش تخصیص')),

)

allocation = models.ForeignKey(BudgetAllocation, on\_delete=models.CASCADE,

related\_name='transactions', verbose\_name=\_("تخصیص بودجه"))

transaction\_type = models.CharField(max\_length=20, choices=TRANSACTION\_TYPES,

verbose\_name=\_("نوع تراکنش"))

amount = models.DecimalField(max\_digits=25, decimal\_places=2, verbose\_name=\_("مبلغ"))

related\_tankhah = models.ForeignKey('tankhah.Tankhah', on\_delete=models.SET\_NULL, null=True, blank=True,

verbose\_name=\_("تنخواه مرتبط"))

timestamp = models.DateTimeField(auto\_now\_add=True, verbose\_name=\_("زمان"))

created\_by = models.ForeignKey('accounts.CustomUser', on\_delete=models.SET\_NULL, null=True,

verbose\_name=\_("کاربر"))

description = models.TextField(blank=True, verbose\_name=\_("توضیحات"))

transaction\_id = models.CharField(max\_length=50, unique=True, verbose\_name=\_("شناسه تراکنش"))#یک شناسه منحصربه‌فرد برای هر تراکنش:

def save(self, \*args, \*\*kwargs):

from django.core.exceptions import ValidationError

from django.contrib.contenttypes.models import ContentType

if not self.transaction\_id:

self.transaction\_id = f"TX-{self.allocation.id}-{timezone.now().timestamp()}"

if self.transaction\_type == 'CONSUMPTION':

if self.amount > self.allocation.remaining\_amount:

raise ValidationError(\_("مبلغ مصرف بیشتر از باقی‌مانده تخصیص است."))

elif self.transaction\_type == 'RETURN':

if self.amount > self.allocation.allocated\_amount:

raise ValidationError(\_("مبلغ برگشتی نمی‌تواند بیشتر از مبلغ تخصیص‌یافته باشد."))

if not self.allocation.budget\_period.is\_active:

raise ValidationError(\_("نمی‌توان از دوره غیرفعال بودجه برگشت داد."))

self.allocation.remaining\_amount += self.amount # remaining\_amount تخصیص افزایش یابد.

self.allocation.allocated\_amount -= self.amount # کاهش مبلغ تخصیص

self.allocation.returned\_amount += self.amount

self.allocation.budget\_period.total\_allocated -= self.amount

self.allocation.budget\_period.returned\_amount += self.amount

self.allocation.save(update\_fields=['remaining\_amount', 'allocated\_amount', 'returned\_amount'])

self.allocation.budget\_period.save(update\_fields=['total\_allocated', 'returned\_amount'])

# self.allocation.budget\_period.save(update\_fields=['total\_allocated'])

# ثبت در BudgetHistory

BudgetHistory.objects.create(

content\_type=ContentType.objects.get\_for\_model(BudgetAllocation),

object\_id=self.allocation.id,

action='RETURN',

amount=self.amount,

created\_by=self.created\_by,

details=f"برگشت {self.amount:,} از تخصیص {self.allocation.id} به دوره بودجه {self.allocation.budget\_period.name}",

transaction\_type='RETURN',

transaction\_id=f"RET-{self.transaction\_id}"

)

elif self.transaction\_type in ['CONSUMPTION', 'ADJUSTMENT\_DECREASE']:

self.allocation.remaining\_amount -= self.amount

elif self.transaction\_type in ['ALLOCATION', 'ADJUSTMENT\_INCREASE']:

self.allocation.remaining\_amount += self.amount

self.allocation.save(update\_fields=['remaining\_amount', 'allocated\_amount'])

super().save(\*args, \*\*kwargs)

def \_\_str\_\_(self):

return f"{self.transaction\_type} - {self.amount:,} ({self.timestamp})"

class Meta:

verbose\_name = \_("تراکنش بودجه")

verbose\_name\_plural = \_("تراکنش‌های بودجه")

default\_permissions = ()

permissions = [

('BudgetTransaction\_add', 'افزودن تراکنش بودجه'),

('BudgetTransaction\_update', 'بروزرسانی تراکنش بودجه'),

('BudgetTransaction\_delete', 'حــذف تراکنش بودجه'),

('BudgetTransaction\_view', \_('نمایش تراکنش بودجه')),

('BudgetTransaction\_return', \_('برگشت تراکنش بودجه')), # جدید

# add/update/delete معمولاً توسط سیستم انجام می‌شه، نه کاربر

]

"""PaymentOrder (دستور پرداخت):"""

class PaymentOrder(models.Model):

"""توضیح: مدل جدا برای دستور پرداخت با لینک به تنخواه و فاکتورها. min\_signatures برای کنترل تعداد امضا و شماره‌گذاری پویا با تاریخ شمسی."""

STATUS\_CHOICES = (

('DRAFT', \_('پیش‌نویس')),

('PENDING\_SIGNATURES', \_('در انتظار امضا')),

('ISSUED', \_('صادر شده')),

('PAID', \_('پرداخت شده')),

('CANCELED', \_('لغو شده')),

)

tankhah = models.ForeignKey('tankhah.Tankhah', on\_delete=models.CASCADE,

related\_name='payment\_orders', verbose\_name=\_("تنخواه"))

order\_number = models.CharField(max\_length=50, unique=True, blank=True,

verbose\_name=\_("شماره دستور پرداخت"))

issue\_date = models.DateField(default=timezone.now, verbose\_name=\_("تاریخ صدور"))

amount = models.DecimalField(max\_digits=25, decimal\_places=2, verbose\_name=\_("مبلغ"))

payee = models.ForeignKey('Payee', on\_delete=models.SET\_NULL, null=True,

verbose\_name=\_("دریافت‌کننده"))

description = models.TextField(verbose\_name=\_("شرح پرداخت"))

related\_factors = models.ManyToManyField('tankhah.Factor', blank=True,

verbose\_name=\_("فاکتورهای مرتبط"))

payment\_id = models.CharField(max\_length=50, blank=True, null=True,

verbose\_name=\_("شناسه پرداخت"))

status = models.CharField(max\_length=20, choices=STATUS\_CHOICES, default='DRAFT',

verbose\_name=\_("وضعیت"))

created\_by\_post = models.ForeignKey('core.Post', on\_delete=models.SET\_NULL, null=True,

verbose\_name=\_("پست ایجادکننده"))

min\_signatures = models.IntegerField(default=1, verbose\_name=\_("حداقل تعداد امضا"))

created\_by = models.ForeignKey('accounts.CustomUser', on\_delete=models.SET\_NULL, null=True,

related\_name='payment\_orders\_created', verbose\_name=\_("ایجادکننده"))

payment\_date = models.DateField(null=True, blank=True, verbose\_name=\_("تاریخ پرداخت"))

def generate\_order\_number(self):

sep = "-"

date\_str = jdatetime.date.fromgregorian(date=self.issue\_date).strftime('%Y%m%d')

serial = PaymentOrder.objects.filter(issue\_date=self.issue\_date).count() + 1

return f"PO{sep}{date\_str}{sep}{serial:03d}"

def save(self, \*args, \*\*kwargs):

if not self.order\_number:

self.order\_number = self.generate\_order\_number()

super().save(\*args, \*\*kwargs)

def \_\_str\_\_(self):

return f"{self.order\_number} - {self.amount}"

class Meta:

verbose\_name = \_("دستور پرداخت")

verbose\_name\_plural = \_("دستورهای پرداخت")

default\_permissions = ()

permissions = [

('PaymentOrder\_add', \_('افزودن دستور پرداخت')),

('PaymentOrder\_view', \_('نمایش دستور پرداخت')),

('PaymentOrder\_update', \_('بروزرسانی دستور پرداخت')),

('PaymentOrder\_delete', \_('حذف دستور پرداخت')),

('PaymentOrder\_sign', \_('امضای دستور پرداخت')),

('PaymentOrder\_issue', \_('صدور دستور پرداخت')),

]

"""Payee (دریافت‌کننده):"""

class Payee(models.Model):

"""توضیح: اطلاعات دریافت‌کننده پرداخت با نوع مشخص (فروشنده، کارمند، دیگر)."""

PAYEE\_TYPES = (

('VENDOR', \_('فروشنده')),

('EMPLOYEE', \_('کارمند')),

('OTHER', \_('دیگر')),

)

name = models.CharField(max\_length=100, verbose\_name=\_("نام"))

family = models.CharField(max\_length=100, verbose\_name=\_("نام خانوادگی"))

payee\_type = models.CharField(max\_length=20, choices=PAYEE\_TYPES, verbose\_name=\_("نوع"))

national\_id = models.CharField(max\_length=20, blank=True, null=True,

verbose\_name=\_("کد ملی/اقتصادی"))

account\_number = models.CharField(max\_length=50, blank=True, null=True,

verbose\_name=\_("شماره حساب"))

iban = models.CharField(max\_length=34, blank=True, null=True, verbose\_name=\_("شبا"))

address = models.TextField(blank=True, null=True, verbose\_name=\_("آدرس"))

phone = models.CharField(max\_length=20, blank=True, null=True, verbose\_name=\_("تلفن"))

created\_by = models.ForeignKey('accounts.CustomUser', on\_delete=models.SET\_NULL, null=True,

related\_name='payees\_created', verbose\_name=\_("ایجادکننده"))

is\_active= models.BooleanField(verbose\_name=\_('فعال'))

def \_\_str\_\_(self):

return f"{self.name} ({self.payee\_type})"

class Meta:

verbose\_name = \_("دریافت‌کننده")

verbose\_name\_plural = \_("دریافت‌کنندگان")

default\_permissions = ()

permissions = [

('Payee\_add', \_('افزودن دریافت‌کننده')),

('Payee\_view', \_('نمایش دریافت‌کننده')),

('Payee\_update', \_('بروزرسانی دریافت‌کننده')),

('Payee\_delete', \_('حذف دریافت‌کننده')),

]

"""TransactionType (نوع تراکنش):"""

class TransactionType(models.Model):

"""توضیح: تعریف پویا نوع تراکنش‌ها (مثل بیمه، جریمه) با امکان نیاز به تأیید اضافی."""

name = models.CharField(max\_length=250, unique=True, verbose\_name=\_("نام"))

description = models.TextField(blank=True, verbose\_name=\_("توضیحات"))

requires\_extra\_approval = models.BooleanField(default=False,

verbose\_name=\_("نیاز به تأیید اضافی"))

created\_by = models.ForeignKey('accounts.CustomUser', on\_delete=models.SET\_NULL, null=True,

related\_name='transaction\_types\_created', verbose\_name=\_("ایجادکننده"))

category = models.CharField(max\_length=50, blank=True, null=True,verbose\_name=\_('گروه بندی تراکنش‌ها '))

def \_\_str\_\_(self):

return self.name

class Meta:

verbose\_name = \_("نوع تراکنش")

verbose\_name\_plural = \_("انواع تراکنش")

default\_permissions = ()

permissions = [

('TransactionType\_add', \_('افزودن نوع تراکنش')),

('TransactionType\_view', \_('نمایش نوع تراکنش')),

('TransactionType\_update', \_('بروزرسانی نوع تراکنش')),

('TransactionType\_delete', \_('حذف نوع تراکنش')),

]

"""تخصیص بودجه به پروژه و زیر پروژه """

class ProjectBudgetAllocation(models.Model):

"""

تخصیص بودجه از BudgetAllocation شعبه به پروژه یا زیرپروژه.

توضیح: این مدل بودجه شعبه رو به پروژه یا زیرپروژه وصل می‌کنه و تاریخچه تخصیص رو نگه می‌داره.

"""

budget\_allocation = models.ForeignKey(

'budgets.BudgetAllocation', on\_delete=models.CASCADE, related\_name='project\_allocations',

verbose\_name=\_("تخصیص بودجه شعبه"))

project = models.ForeignKey('core.Project', on\_delete=models.CASCADE, related\_name='budget\_allocations',

verbose\_name=\_("پروژه"))

subproject = models.ForeignKey('core.SubProject', on\_delete=models.CASCADE, null=True, blank=True,

related\_name='budget\_allocations', verbose\_name=\_("زیرپروژه"))

allocated\_amount = models.DecimalField(max\_digits=25, decimal\_places=2, verbose\_name=\_("مبلغ تخصیص"))

allocation\_date = models.DateField(default=timezone.now, verbose\_name=\_("تاریخ تخصیص"))

created\_by = models.ForeignKey('accounts.CustomUser', on\_delete=models.SET\_NULL, null=True,

related\_name='project\_budget\_allocations\_created', verbose\_name=\_("ایجادکننده"))

description = models.TextField(blank=True, verbose\_name=\_("توضیحات"))

def get\_remaining\_amount(self):

from budgets.budget\_calculations import get\_subproject\_remaining\_budget

if self.subproject:

return get\_subproject\_remaining\_budget(self.subproject)

from budgets.budget\_calculations import get\_project\_remaining\_budget

return get\_project\_remaining\_budget(self.project)

def clean(self):

remaining = self.budget\_allocation.get\_remaining\_amount()

if self.allocated\_amount > remaining:

raise ValidationError(\_("مبلغ تخصیص بیشتر از بودجه باقی‌مانده شعبه است"))

if self.subproject and self.subproject.project != self.project:

raise ValidationError(\_("زیرپروژه باید به پروژه انتخاب‌شده تعلق داشته باشد"))

if self.allocated\_amount < 0:

raise ValidationError(\_("مبلغ تخصیص نمی‌تواند منفی باشد"))

if self.budget\_allocation and self.allocation\_date:

start\_date = self.budget\_allocation.budget\_period.start\_date

end\_date = self.budget\_allocation.budget\_period.end\_date

if not (start\_date <= self.allocation\_date <= end\_date):

raise ValidationError(\_("تاریخ تخصیص باید در بازه دوره بودجه باشد"))

def save(self, \*args, \*\*kwargs):

self.clean()

if not self.pk:

self.remaining\_amount = self.allocated\_amount

super().save(\*args, \*\*kwargs)

self.budget\_allocation.remaining\_amount = self.budget\_allocation.get\_remaining\_amount()

self.budget\_allocation.save(update\_fields=['remaining\_amount'])

self.remaining\_amount = self.get\_remaining\_amount()

super().save(update\_fields=['remaining\_amount'])

def \_\_str\_\_(self):

target = self.subproject.name if self.subproject else self.project.name

return f"{target} - {self.allocated\_amount:,} ({self.allocation\_date})"

class Meta:

verbose\_name = \_("تخصیص بودجه پروژه")

verbose\_name\_plural = \_("تخصیص‌های بودجه پروژه")

default\_permissions = ()

permissions = [

('ProjectBudgetAllocation\_add', \_('افزودن تخصیص بودجه پروژه')),

('ProjectBudgetAllocation\_view', \_('نمایش تخصیص بودجه پروژه')),

('ProjectBudgetAllocation\_update', \_('بروزرسانی تخصیص بودجه پروژه')),

('ProjectBudgetAllocation\_delete', \_('حذف تخصیص بودجه پروژه')),

('ProjectBudgetAllocation\_Head\_Office', 'تخصیص بودجه مجموعه پروژه(دفتر مرکزی)🏠'),

('ProjectBudgetAllocation\_Branch', 'تخصیص بودجه مجموعه پروژه(شعبه)🏠'),

]

"""مدل BudgetReallocation برای انتقال باقی‌مانده بودجه متوقف‌شده."""

class BudgetReallocation(models.Model):

source\_allocation = models.ForeignKey('BudgetAllocation', on\_delete=models.CASCADE, related\_name='reallocations\_from')

target\_allocation = models.ForeignKey('BudgetAllocation', on\_delete=models.CASCADE, null=True, related\_name='reallocations\_to')

amount = models.DecimalField(max\_digits=25, decimal\_places=2)

reallocation\_date = models.DateField(default=timezone.now)

reason = models.TextField()

created\_by = models.ForeignKey(CustomUser, on\_delete=models.SET\_NULL, null=True)

"""یه جدول تنظیمات (BudgetSettings) برای مدیریت قفل و هشدار در سطوح مختلف:"""

class BudgetSettings(models.Model):

level = models.CharField(max\_length=50, choices=[('PERIOD', 'دوره بودجه'), ('ALLOCATION', 'تخصیص'), ('PROJECT', 'پروژه')])

locked\_percentage = models.DecimalField(max\_digits=5, decimal\_places=2)

warning\_threshold = models.DecimalField(max\_digits=5, decimal\_places=2)

warning\_action = models.CharField(max\_length=50, choices=[('NOTIFY', 'اعلان'), ('LOCK', 'قفل'), ('RESTRICT', 'محدود')])

organization = models.ForeignKey('core.Organization', on\_delete=models.CASCADE, null=True)

budget\_period = models.ForeignKey('BudgetPeriod', on\_delete=models.CASCADE, null=True, verbose\_name=\_("دوره بودجه"))

# ردیابی مدل

# content\_type = models.ForeignKey(ContentType, on\_delete=models.CASCADE)

# object\_id = models.PositiveIntegerField()

# content\_object = GenericForeignKey('content\_type', 'object\_id')

"""مدل BudgetHistory برای لاگ کردن تغییرات بودجه و تخصیص‌ها:"""

# ------------------------------------

"""تاریخچه برای هر بودجه کلان"""

class BudgetHistory(models.Model):

content\_type = models.ForeignKey(ContentType, on\_delete=models.CASCADE)

object\_id = models.PositiveIntegerField()

# content\_object = GenericForeignKey('content\_type', 'object\_id')

action = models.CharField(max\_length=50, choices=[

('CREATE', \_('ایجاد')),

('UPDATE', \_('بروزرسانی')),

('STOP', \_('توقف')),

('REALLOCATE', \_('انتقال')),

('RETURN', \_('برگشت')) # جدید

])

amount = models.DecimalField(max\_digits=25, decimal\_places=2, null=True, verbose\_name=\_('رقم عدد'))

created\_by = models.ForeignKey(CustomUser, on\_delete=models.SET\_NULL, null=True, verbose\_name=\_('کاربر ثبت کننده'))

created\_at = models.DateTimeField(auto\_now\_add=True, verbose\_name=\_('اکشن در تاریخ و ساعت'))

details = models.TextField(blank=True, verbose\_name=\_('جزئیات'))

transaction\_type = models.CharField(max\_length=20, choices=[('ALLOCATION', \_('تخصیص')), ('CONSUMPTION', \_('مصرف')), ('RETURN', \_('برگشت'))],verbose\_name=\_("نوع تراکنش"))

transaction\_id = models.CharField(max\_length=50, unique=True, verbose\_name=\_("شناسه تراکنش"))

def \_\_str\_\_(self):

return f"{self.action} - {self.amount:,} ({self.created\_at})"

class Meta:

verbose\_name = \_("تاریخچه بودجه")

verbose\_name\_plural = \_("تاریخچه‌های بودجه")

default\_permissions = ()

permissions = [

('BudgetHistory\_add','افزودن تاریخچه برای هر بودجه کلان'),

('BudgetHistory\_view',' نمایش تاریخچه هر بودجه کلان'),

('BudgetHistory\_update','بروزرسانی تاریخچه برای هر بودجه کلان'),

('BudgetHistory\_delete',' حــذف تاریخچه برای هر بودجه کلان'),

]

class SystemSettings(models.Model):

"""تنظیمات سیستم بودجه"""

budget\_locked\_percentage\_default = models.DecimalField(

max\_digits=5, decimal\_places=2, default=0, verbose\_name=\_("درصد قفل‌شده پیش‌فرض بودجه")

)

budget\_warning\_threshold\_default = models.DecimalField(

max\_digits=5, decimal\_places=2, default=10, verbose\_name=\_("آستانه هشدار پیش‌فرض بودجه")

)

budget\_warning\_action\_default = models.CharField(

max\_length=50, choices=[('NOTIFY', 'اعلان'), ('LOCK', 'قفل'), ('RESTRICT', 'محدود')],

default='NOTIFY', verbose\_name=\_("اقدام هشدار پیش‌فرض بودجه")

)

allocation\_locked\_percentage\_default = models.DecimalField(

max\_digits=5, decimal\_places=2, default=0, verbose\_name=\_("درصد قفل‌شده پیش‌فرض تخصیص")

)

class Meta:

verbose\_name = \_("تنظیمات سیستم")

verbose\_name\_plural = \_("تنظیمات سیستم")

def \_\_str\_\_(self):

return "تنظیمات سیستم بودجه"

میخوام مدل رو بررسی کنی

import logging

from django import forms

from django.utils.translation import gettext\_lazy as \_

from django.utils import timezone

from decimal import Decimal

import jdatetime

from Tanbakhsystem.utils import format\_jalali\_date, parse\_jalali\_date

from budgets.models import BudgetPeriod, SystemSettings, BudgetTransaction, BudgetAllocation

from core.templatetags.rcms\_custom\_filters import number\_to\_farsi\_words

logger = logging.getLogger(\_\_name\_\_)

class BudgetReturnForm(forms.ModelForm):

""" فرم برای ثبت تراکنش برگشت:"""

class Meta:

model = BudgetTransaction

fields = ['allocation', 'amount', 'description']

widgets = {

'allocation': forms.Select(attrs={'class': 'form-control'}),

'amount': forms.NumberInput(attrs={'class': 'form-control', 'min': 0}),

'description': forms.Textarea(attrs={'class': 'form-control', 'rows': 1}),

}

labels = {

'allocation': \_('تخصیص بودجه'),

'amount': \_('مبلغ برگشتی'),

'description': \_('توضیحات'),

}

def \_\_init\_\_(self, \*args, \*\*kwargs):

user = kwargs.pop('user', None)

super().\_\_init\_\_(\*args, \*\*kwargs)

self.fields['allocation'].queryset = BudgetAllocation.objects.filter(is\_active=True)

self.instance.transaction\_type = 'RETURN'

# self.instance.created\_by = kwargs.get('user')

if user:

self.instance.created\_by = user

def clean\_amount(self):

amount = self.cleaned\_data.get('amount')

allocation = self.cleaned\_data.get('allocation')

if amount <= 0:

raise forms.ValidationError(\_('مبلغ برگشتی باید مثبت باشد.'))

if amount > allocation.allocated\_amount:

raise forms.ValidationError(\_('مبلغ برگشتی نمی‌تواند بیشتر از تخصیص باشد.'))

return amount

def save(self, commit=True):

instance = super().save(commit=False)

instance.transaction\_type = 'RETURN'

if commit:

instance.save()

# ارسال اعلان

status, message = instance.allocation.check\_allocation\_status()

if status in ('warning', 'locked', 'completed', 'stopped'):

instance.allocation.send\_notification(status, message)

return instance

from django.contrib.auth.mixins import LoginRequiredMixin

from django.views.generic import CreateView

from django.urls import reverse\_lazy

from django.contrib import messages

class BudgetReturnView(LoginRequiredMixin, CreateView):

from budgets.models import BudgetTransaction

from budgets.BudgetReturn.forms\_BudgetReturm import BudgetReturnForm

model = BudgetTransaction

form\_class = BudgetReturnForm

template\_name = 'budgets/budget/budget\_return\_form.html'

success\_url = reverse\_lazy('budgetallocation\_list')

def get\_form\_kwargs(self):

kwargs = super().get\_form\_kwargs()

kwargs['user'] = self.request.user

return kwargs

def form\_valid(self, form):

messages.success(self.request, \_('بودجه با موفقیت برگشت داده شد.'))

return super().form\_valid(form)

def form\_invalid(self, form):

messages.error(self.request, \_('خطایی در برگشت بودجه رخ داد.'))

return super().form\_invalid(form)

""""گزارش‌گیری بودجه برگشتی:"""

from budgets.models import BudgetHistory

def get\_returned\_budgets(budget\_period):

from django.contrib.contenttypes.models import ContentType

from budgets.models import BudgetAllocation

return BudgetHistory.objects.filter(

content\_type=ContentType.objects.get\_for\_model(BudgetAllocation),

action='RETURN',

content\_object\_\_budget\_period=budget\_period

).values('amount', 'details', 'created\_at', 'created\_by\_\_username')

import logging

from django import forms

from django.utils.translation import gettext\_lazy as \_

from django.utils import timezone

from decimal import Decimal

import jdatetime

from Tanbakhsystem.utils import format\_jalali\_date, parse\_jalali\_date

from budgets.models import BudgetPeriod, SystemSettings

from core.templatetags.rcms\_custom\_filters import number\_to\_farsi\_words

logger = logging.getLogger(\_\_name\_\_)

def to\_english\_digits(text):

"""تبدیل اعداد فارسی به انگلیسی"""

farsi\_to\_english = str.maketrans('۰۱۲۳۴۵۶۷۸۹', '0123456789')

return str(text).translate(farsi\_to\_english)

class BudgetPeriodForm(forms.ModelForm):

start\_date = forms.CharField(

label=\_('تاریخ شروع'),

widget=forms.TextInput(attrs={

'data-jdp': '',

'class': 'form-control jalali-datepicker',

'autocomplete': 'off'

}),

required=True

)

end\_date = forms.CharField(

label=\_('تاریخ پایان'),

widget=forms.TextInput(attrs={

'data-jdp': '',

'class': 'form-control jalali-datepicker',

'autocomplete': 'off'

}),

required=True

)

class Meta:

model = BudgetPeriod

fields = [

'organization', 'name', 'start\_date', 'end\_date', 'total\_amount',

'is\_active', 'is\_archived', 'is\_completed', 'lock\_condition',

'locked\_percentage', 'warning\_threshold', 'warning\_action', 'description'

]

widgets = {

'organization': forms.Select(attrs={'class': 'form-control', 'required': True}),

'name': forms.TextInput(attrs={'class': 'form-control', 'placeholder': \_('نام دوره (مثل بودجه ۱۴۰۴)')}),

'total\_amount': forms.NumberInput(attrs={'class': 'form-control', 'min': 1}),

'is\_active': forms.CheckboxInput(attrs={'class': 'form-check-input'}),

'is\_archived': forms.CheckboxInput(attrs={'class': 'form-check-input'}),

'is\_completed': forms.CheckboxInput(attrs={'class': 'form-check-input'}),

'lock\_condition': forms.Select(attrs={'class': 'form-control'}),

'locked\_percentage': forms.NumberInput(attrs={'class': 'form-control', 'min': 0, 'max': 100, 'step': 0.01}),

'warning\_threshold': forms.NumberInput(attrs={'class': 'form-control', 'min': 0, 'max': 100, 'step': 0.01}),

'warning\_action': forms.Select(attrs={'class': 'form-control'}),

'description': forms.Textarea(attrs={'class': 'form-control', 'rows': 1}),

}

labels = {

'organization': \_('دفتر مرکزی'),

'name': \_('نام دوره بودجه'),

'total\_amount': \_('مبلغ کل (ریال)'),

'is\_active': \_('فعال'),

'is\_archived': \_('بایگانی شده'),

'is\_completed': \_('تمام‌شده'),

'lock\_condition': \_('شرط قفل'),

'locked\_percentage': \_('درصد قفل‌شده'),

'warning\_threshold': \_('آستانه اخطار'),

'warning\_action': \_('اقدام هشدار'),

}

def \_\_init\_\_(self, \*args, \*\*kwargs):

self.user = kwargs.pop('user', None)

super().\_\_init\_\_(\*args, \*\*kwargs)

logger.debug("Initializing BudgetPeriodForm")

# محدود کردن سازمان‌ها به دفاتر مرکزی فعال

from core.models import Organization

self.fields['organization'].queryset = Organization.objects.filter(is\_core=True, is\_active=True)

# تنظیم مقادیر پیش‌فرض از SystemSettings

system\_settings = SystemSettings.objects.first()

if system\_settings:

self.fields['locked\_percentage'].initial = system\_settings.budget\_locked\_percentage\_default

self.fields['warning\_threshold'].initial = system\_settings.budget\_warning\_threshold\_default

self.fields['warning\_action'].initial = system\_settings.budget\_warning\_action\_default

# تنظیم تاریخ‌های اولیه

if self.instance and self.instance.pk:

self.initial['start\_date'] = format\_jalali\_date(self.instance.start\_date)

self.initial['end\_date'] = format\_jalali\_date(self.instance.end\_date)

logger.debug(f"Set initial dates: start\_date={self.initial['start\_date']}, end\_date={self.initial['end\_date']}")

else:

current\_jalali\_year = jdatetime.date.today().year

self.initial['start\_date'] = f"{current\_jalali\_year}/01/01"

self.initial['end\_date'] = f"{current\_jalali\_year}/12/29"

# مقدار حروف برای total\_amount

self.total\_amount\_words = ""

if self.instance and self.instance.total\_amount:

self.total\_amount\_words = number\_to\_farsi\_words(self.instance.total\_amount)

logger.info(f' self.total\_amount\_words IS {self.total\_amount\_words }')

def clean\_start\_date(self):

date\_str = self.cleaned\_data.get('start\_date')

logger.debug(f"Cleaning start\_date: input={date\_str}")

if not date\_str:

logger.error("start\_date is empty")

raise forms.ValidationError(\_('تاریخ شروع اجباری است.'))

try:

date\_str = to\_english\_digits(date\_str)

parsed\_date = parse\_jalali\_date(date\_str, field\_name=\_('تاریخ شروع'))

logger.debug(f"Parsed start\_date: {parsed\_date}")

return parsed\_date

except Exception as e:

logger.error(f"Error parsing start\_date: {str(e)}")

raise forms.ValidationError(\_('فرمت تاریخ شروع نامعتبر است.'))

def clean\_end\_date(self):

date\_str = self.cleaned\_data.get('end\_date')

logger.debug(f"Cleaning end\_date: input={date\_str}")

if not date\_str:

logger.error("end\_date is empty")

raise forms.ValidationError(\_('تاریخ پایان اجباری است.'))

try:

date\_str = to\_english\_digits(date\_str)

parsed\_date = parse\_jalali\_date(date\_str, field\_name=\_('تاریخ پایان'))

logger.debug(f"Parsed end\_date: {parsed\_date}")

return parsed\_date

except Exception as e:

logger.error(f"Error parsing end\_date: {str(e)}")

raise forms.ValidationError(\_('فرمت تاریخ پایان نامعتبر است.'))

def clean\_total\_amount(self):

amount = self.cleaned\_data.get('total\_amount')

logger.debug(f"Cleaning total\_amount: input={amount}")

if amount is None:

logger.error("total\_amount is None")

raise forms.ValidationError(\_('مبلغ کل نمی‌تواند خالی باشد.'))

if amount <= 0:

logger.error(f"Invalid total\_amount: {amount} (must be positive)")

raise forms.ValidationError(\_('مبلغ کل باید بزرگ‌تر از صفر باشد.'))

# به‌روزرسانی مقدار حروف

self.total\_amount\_words = number\_to\_farsi\_words(amount)

logger.debug(f"Validated total\_amount: {amount}, words: {self.total\_amount\_words}")

return amount

def clean\_locked\_percentage(self):

percentage = self.cleaned\_data.get('locked\_percentage')

logger.debug(f"Cleaning locked\_percentage: input={percentage}")

if percentage is None:

logger.error("locked\_percentage is None")

raise forms.ValidationError(\_('درصد قفل‌شده نمی‌تواند خالی باشد.'))

if not (0 <= percentage <= 100):

logger.error(f"Invalid locked\_percentage: {percentage} (must be 0-100)")

raise forms.ValidationError(\_('درصد قفل‌شده باید بین ۰ تا ۱۰۰ باشد.'))

logger.debug(f"Validated locked\_percentage: {percentage}")

return percentage

def clean\_warning\_threshold(self):

threshold = self.cleaned\_data.get('warning\_threshold')

logger.debug(f"Cleaning warning\_threshold: input={threshold}")

if threshold is None:

logger.error("warning\_threshold is None")

raise forms.ValidationError(\_('آستانه اخطار نمی‌تواند خالی باشد.'))

if not (0 <= threshold <= 100):

logger.error(f"Invalid warning\_threshold: {threshold} (must be 0-100)")

raise forms.ValidationError(\_('آستانه اخطار باید بین ۰ تا ۱۰۰ باشد.'))

locked\_percentage = self.cleaned\_data.get('locked\_percentage')

if locked\_percentage is not None and threshold <= locked\_percentage:

logger.error(f"warning\_threshold ({threshold}) <= locked\_percentage ({locked\_percentage})")

raise forms.ValidationError(\_('آستانه اخطار باید بزرگ‌تر از درصد قفل‌شده باشد.'))

logger.debug(f"Validated warning\_threshold: {threshold}")

return threshold

def clean\_organization(self):

organization = self.cleaned\_data.get('organization')

logger.debug(f"Cleaning organization: input={organization}")

if organization and not organization.is\_core:

logger.error(f"Selected organization {organization} is not core")

raise forms.ValidationError(\_('فقط دفاتر مرکزی می‌توانند برای بودجه کلان انتخاب شوند.'))

return organization

def clean(self):

cleaned\_data = super().clean()

logger.debug(f"Running clean method: cleaned\_data={cleaned\_data}")

start\_date = cleaned\_data.get('start\_date')

end\_date = cleaned\_data.get('end\_date')

is\_completed = cleaned\_data.get('is\_completed')

is\_active = cleaned\_data.get('is\_active')

organization = cleaned\_data.get('organization')

name = cleaned\_data.get('name')

if start\_date and end\_date:

if end\_date <= start\_date:

logger.error(f"end\_date ({end\_date}) <= start\_date ({start\_date})")

raise forms.ValidationError(\_('تاریخ پایان باید بعد از تاریخ شروع باشد.'))

overlapping\_periods = BudgetPeriod.objects.filter(

organization=organization,

start\_date\_\_lte=end\_date,

end\_date\_\_gte=start\_date

).exclude(pk=self.instance.pk)

if overlapping\_periods.exists():

logger.error("Overlapping budget period detected")

raise forms.ValidationError(\_('دوره بودجه با تاریخ‌های متداخل در این سازمان وجود دارد.'))

logger.debug("Validated date comparison")

else:

logger.warning(f"Missing dates: start\_date={start\_date}, end\_date={end\_date}")

if is\_completed and is\_active:

logger.error("is\_completed and is\_active both True")

raise forms.ValidationError(\_('دوره تمام‌شده نمی‌تواند فعال باشد.'))

if organization and name:

if BudgetPeriod.objects.filter(organization=organization, name=name).exclude(pk=self.instance.pk).exists():

logger.error(f"Duplicate name '{name}' for organization {organization}")

raise forms.ValidationError(\_('نام دوره در این سازمان قبلاً استفاده شده است.'))

logger.debug("Validated name uniqueness")

logger.debug("Clean method completed successfully")

return cleaned\_data

def save(self, commit=True):

logger.debug("Saving BudgetPeriodForm")

instance = super().save(commit=False)

if not self.user or not self.user.is\_authenticated:

logger.error("No authenticated user provided")

raise forms.ValidationError(\_('کاربر معتبر برای ایجاد دوره بودجه لازم است.'))

instance.created\_by = self.user

logger.debug(f"Set created\_by: {instance.created\_by}")

if commit:

try:

instance.save()

logger.info(f"BudgetPeriod saved: {instance}")

except Exception as e:

logger.error(f"Error saving BudgetPeriod: {str(e)}")

raise

return instance

from decimal import Decimal

from django.db.models import Q, Sum

from django.shortcuts import redirect

from django.views.generic import CreateView, UpdateView, DeleteView,ListView,DetailView

from django.urls import reverse\_lazy

from django.contrib import messages

from django.db import transaction

from Tanbakhsystem.utils import parse\_jalali\_date\_jdate

from budgets.budget\_calculations import get\_budget\_details, check\_budget\_status, calculate\_remaining\_budget

from core.PermissionBase import PermissionBaseView

from budgets.models import BudgetPeriod, BudgetTransaction, BudgetHistory

from budgets.BudgetPeriod.Forms\_BudgetPeriod import BudgetPeriodForm

from django.utils.translation import gettext\_lazy as \_

import logging

logger = logging.getLogger(\_\_name\_\_)

class BudgetPeriodCreateView(PermissionBaseView,CreateView):

model = BudgetPeriod

form\_class = BudgetPeriodForm

template\_name = 'budgets/budget/budgetperiod\_form.html'

success\_url = reverse\_lazy('budgetperiod\_list')

def get\_form\_kwargs(self):

kwargs = super().get\_form\_kwargs()

kwargs['initial']['created\_by'] = self.request.user

kwargs['user'] = self.request.user

return kwargs

def get\_context\_data(self, \*\*kwargs):

context = super().get\_context\_data(\*\*kwargs)

context['title'] = \_('ایجاد دوره بودجه جدید')

return context

@transaction.atomic

def form\_valid(self, form):

try:

response = super().form\_valid(form)

messages.success(self.request, \_('دوره بودجه با موفقیت ایجاد شد.'))

return response

except Exception as e:

messages.error(self.request, \_('خطایی در ایجاد دوره بودجه رخ داد: ') + str(e))

return self.form\_invalid(form)

def form\_invalid(self, form):

messages.error(self.request, \_('لطفاً خطاهای فرم را بررسی کنید.'))

return self.render\_to\_response(self.get\_context\_data(form=form))

class BudgetPeriodUpdateView(PermissionBaseView,UpdateView):

model = BudgetPeriod

form\_class = BudgetPeriodForm

template\_name = 'budgets/budget/budgetperiod\_form.html'

success\_url = reverse\_lazy('budgetperiod\_list')

def get\_form\_kwargs(self):

kwargs = super().get\_form\_kwargs()

kwargs['user'] = self.request.user

return kwargs

def get\_context\_data(self, \*\*kwargs):

context = super().get\_context\_data(\*\*kwargs)

context['title'] = \_('ویرایش دوره بودجه')

context['total\_allocated'] = self.object.total\_allocated

context['remaining\_amount'] = self.object.get\_remaining\_amount()

return context

@transaction.atomic

def form\_valid(self, form):

try:

response = super().form\_valid(form)

messages.success(self.request, \_('دوره بودجه با موفقیت به‌روزرسانی شد.'))

return response

except Exception as e:

messages.error(self.request, \_('خطایی در به‌روزرسانی دوره بودجه رخ داد: ') + str(e))

return self.form\_invalid(form)

def form\_invalid(self, form):

messages.error(self.request, \_('لطفاً خطاهای فرم را بررسی کنید.'))

return self.render\_to\_response(self.get\_context\_data(form=form))

class BudgetPeriodDeleteView(PermissionBaseView, DeleteView):

model = BudgetPeriod

template\_name = 'budgets/budget/budgetperiod\_confirm\_delete.html'

success\_url = reverse\_lazy('budgetperiod\_list')

def post(self, request, \*args, \*\*kwargs):

budget\_period = self.get\_object()

with transaction.atomic():

budget\_period.delete()

messages.success(request, f'دوره بودجه {budget\_period.name} با موفقیت حذف شد.')

return redirect(self.success\_url)

# --- BudgetPeriod CRUD ---

class BudgetPeriodListView(PermissionBaseView, ListView):

model = BudgetPeriod

template\_name = 'budgets/budget/budgetperiod\_list.html'

context\_object\_name = 'budget\_periods'

paginate\_by = 10

from Tanbakhsystem.utils import parse\_jalali\_date

def get\_queryset(self):

queryset = super().get\_queryset()

q = self.request.GET.get('q', '')

status = self.request.GET.get('status', '')

date\_from = self.request.GET.get('date\_from', '')

date\_to = self.request.GET.get('date\_to', '')

if q:

queryset = queryset.filter(Q(name\_\_icontains=q) | Q(organization\_\_name\_\_icontains=q))

if status:

status\_map = {

'active': Q(is\_active=True),

'inactive': Q(is\_active=False),

'locked': Q(lock\_condition='MANUAL'),

'completed': Q(is\_completed=True),

}

queryset = queryset.filter(status\_map.get(status, Q()))

try:

if date\_from:

queryset = queryset.filter(start\_date\_\_gte=parse\_jalali\_date\_jdate(date\_from))

if date\_to:

queryset = queryset.filter(end\_date\_\_lte=parse\_jalali\_date\_jdate(date\_to))

except Exception as e:

logger.error(f"Date filtering error: {str(e)}")

return queryset.order\_by('-start\_date')

def get\_context\_data(self, \*\*kwargs):

context = super().get\_context\_data(\*\*kwargs)

queryset = self.get\_queryset() # برای محاسبه status\_summary

for period in context['budget\_periods']:

period.remaining\_amount = max(period.total\_amount - (period.total\_allocated or Decimal('0')), Decimal('0'))

period.status, period.status\_message = check\_budget\_status(period)

if period.status in ('warning', 'locked', 'completed'):

messages.warning(self.request, f"{period.name}: {period.status\_message}")

from django.contrib.contenttypes.models import ContentType

BudgetHistory.objects.create(

content\_type=ContentType.objects.get\_for\_model(BudgetPeriod),

object\_id=period.pk,

action='STATUS\_CHECK',

details=f"وضعیت: {period.status} - {period.status\_message}",

created\_by=self.request.user

)

context['query'] = self.request.GET.get('q', '')

context['status'] = self.request.GET.get('status', '')

context['date\_from'] = self.request.GET.get('date\_from', '')

context['date\_to'] = self.request.GET.get('date\_to', '')

context['status\_summary'] = {

'active': queryset.filter(is\_active=True).count(),

'locked': queryset.filter(lock\_condition='MANUAL').count(),

'completed': queryset.filter(is\_completed=True).count(),

'total': queryset.count(),

}

logger.debug(f"BudgetPeriodListView context: {context}")

return context

class BudgetPeriodDetailView(PermissionBaseView, DetailView):

model = BudgetPeriod

template\_name = 'budgets/budget/budgetperiod\_detail.html'

context\_object\_name = 'budget\_period'

def get\_context\_data(self, \*\*kwargs):

context = super().get\_context\_data(\*\*kwargs)

# صفحه‌بندی تراکنش‌ها

transactions = BudgetTransaction.objects.filter(

allocation\_\_budget\_period=self.object

).order\_by('-timestamp')

from django.core.paginator import Paginator

paginator = Paginator(transactions, 10)

page\_number = self.request.GET.get('page')

context['transactions'] = paginator.get\_page(page\_number)

# جزئیات بودجه

try:

context['budget\_details'] = get\_budget\_details(self.object)

except Exception as e:

logger.error(f"Error in get\_budget\_details: {str(e)}")

messages.error(self.request, \_('خطایی در محاسبه جزئیات بودجه رخ داد.'))

context['budget\_details'] = {

'total\_budget': Decimal('0'),

'total\_allocated': Decimal('0'),

'remaining\_budget': Decimal('0'),

'status': 'error',

'status\_message': 'خطا در محاسبه'

}

# اعلان وضعیت

try:

status\_dict = check\_budget\_status(self.object)

status = status\_dict['status']

message = status\_dict['message']

if status in ('warning', 'locked', 'completed'):

messages.warning(self.request, message)

except Exception as e:

logger.error(f"Error in check\_budget\_status: {str(e)}")

messages.error(self.request, \_('خطایی در بررسی وضعیت بودجه رخ داد.'))

logger.debug(f"BudgetPeriodDetailView context: {context}")

return context

import logging

from time import timezone

from django import forms

from django.core.exceptions import ValidationError

from django.utils.translation import gettext\_lazy as \_

from django.db.models import Sum

from decimal import Decimal

import jdatetime

import re

from budgets.models import BudgetAllocation, BudgetPeriod

from core.models import Organization, Project, OrganizationType

from datetime import datetime, date

logger = logging.getLogger(\_\_name\_\_)

def to\_english\_digits(text):

"""تبدیل اعداد فارسی به انگلیسی"""

farsi\_to\_english = str.maketrans('۰۱۲۳۴۵۶۷۸۹', '0123456789')

return str(text).translate(farsi\_to\_english)

class BudgetAllocationForm(forms.ModelForm):

ALLOCATION\_TYPE\_CHOICES = [

('amount', \_('مبلغ')),

('percent', \_('درصد')),

]

WARNING\_ACTION\_CHOICES = [

('email', \_('ارسال ایمیل هشدار')),

('sms', \_('ارسال پیامک هشدار')),

('notification', \_('ارسال اعلان سیستم')),

]

allocation\_type = forms.ChoiceField(

choices=ALLOCATION\_TYPE\_CHOICES,

widget=forms.RadioSelect(attrs={'class': 'form-check-input'}),

label=\_('نوع تخصیص'),

initial='amount'

)

warning\_action = forms.ChoiceField(

choices=WARNING\_ACTION\_CHOICES,

widget=forms.Select(attrs={'class': 'form-select form-select-lg'}),

label=\_('اقدام هشدار'),

required=False

)

class Meta:

model = BudgetAllocation

fields = [

'budget\_period', 'organization', 'project', 'allocated\_amount',

'allocation\_date', 'description', 'is\_active', 'is\_stopped',

'allocation\_type', 'locked\_percentage', 'warning\_threshold',

'warning\_action', 'allocation\_number'

]

widgets = {

'organization': forms.Select(attrs={

'class': 'form-select form-select-lg',

'required': 'required',

'data-control': 'select2',

'data-placeholder': \_('انتخاب سازمان')

}),

'project': forms.Select(attrs={

'class': 'form-select form-select-lg',

'required': 'required',

'data-control': 'select2',

'data-placeholder': \_('انتخاب پروژه')

}),

'allocated\_amount': forms.NumberInput(attrs={

'class': 'form-control form-control-lg numeric-input',

'min': '0',

'step': '1',

'inputmode': 'numeric',

'required': 'required',

'placeholder': \_('مبلغ را وارد کنید')

}),

'allocation\_date': forms.TextInput(attrs={

'data-jdp': '',

'class': 'form-control form-control-lg',

'placeholder': '1404/01/17',

'required': 'required',

'autocomplete': 'off'

}),

'description': forms.Textarea(attrs={

'class': 'form-control form-control-lg',

'rows': 3,

'placeholder': \_('توضیحات تخصیص بودجه')

}),

'is\_active': forms.CheckboxInput(attrs={

'class': 'form-check-input',

'role': 'switch',

'style': 'width: 2.5rem; height: 1.5rem;'

}),

'is\_stopped': forms.CheckboxInput(attrs={

'class': 'form-check-input',

'role': 'switch',

'style': 'width: 2.5rem; height: 1.5rem;'

}),

'locked\_percentage': forms.NumberInput(attrs={

'class': 'form-control form-control-lg',

'min': '0',

'max': '100',

'step': '1',

'inputmode': 'numeric',

'placeholder': '0-100'

}),

'warning\_threshold': forms.NumberInput(attrs={

'class': 'form-control form-control-lg',

'min': '0',

'max': '100',

'step': '1',

'inputmode': 'numeric',

'placeholder': '0-100'

}),

'allocation\_number': forms.NumberInput(attrs={

'class': 'form-control form-control-lg',

'min': '0',

'step': '1',

'inputmode': 'numeric',

'placeholder': \_('شماره تخصیص')

}),

'budget\_period': forms.HiddenInput(),

}

labels = {

'organization': \_('سازمان'),

'project': \_('پروژه'),

'allocated\_amount': \_('مبلغ تخصیص'),

'allocation\_date': \_('تاریخ تخصیص'),

'description': \_('شرح'),

'is\_active': \_('فعال'),

'is\_stopped': \_('متوقف شده'),

'locked\_percentage': \_('درصد قفل شده'),

'warning\_threshold': \_('آستانه اخطار'),

'allocation\_number': \_('شماره تخصیص'),

}

help\_texts = {

'locked\_percentage': \_('درصدی از بودجه که پس از رسیدن به آن، تخصیص جدید غیرفعال می‌شود'),

'warning\_threshold': \_('درصدی از بودجه که پس از رسیدن به آن، هشدار ارسال می‌شود'),

}

def \_\_init\_\_(self, \*args, \*\*kwargs):

self.budget\_period = kwargs.pop('budget\_period', None)

self.user = kwargs.pop('user', None)

logger.debug(f"Initializing BudgetAllocationForm with budget\_period={self.budget\_period}, user={self.user}")

super().\_\_init\_\_(\*args, \*\*kwargs)

# تنظیم کوئری‌ست برای سازمان‌های قابل تخصیص بودجه

allowed\_org\_types = OrganizationType.objects.filter(

is\_budget\_allocatable=True

).values\_list('id', flat=True)

self.fields['organization'].queryset = Organization.objects.filter(

org\_type\_\_in=allowed\_org\_types,

is\_active=True

).select\_related('org\_type')

# تنظیم کوئری‌ست برای پروژه‌های فعال

# self.fields['project'].queryset = Project.objects.filter(

# is\_active=True

# ).select\_related('category')

if self.budget\_period:

self.fields['budget\_period'].initial = self.budget\_period

# فیلتر سازمان‌هایی که قبلاً برای این دوره بودجه تخصیص داده‌اند

allocated\_orgs = BudgetAllocation.objects.filter(

budget\_period=self.budget\_period

).values\_list('organization\_id', flat=True)

if allocated\_orgs:

self.fields['organization'].queryset = self.fields['organization'].queryset.filter(

id\_\_in=allocated\_orgs

)

logger.debug(

f"Filtered organization queryset: {list(self.fields['organization'].queryset.values('id', 'name'))}")

# تبدیل تاریخ میلادی به شمسی برای نمایش

if self.instance.pk and self.instance.allocation\_date:

j\_date = jdatetime.date.fromgregorian(date=self.instance.allocation\_date)

self.initial['allocation\_date'] = j\_date.strftime('%Y/%m/%d')

def clean\_allocation\_date(self):

date\_input = self.cleaned\_data.get('allocation\_date')

logger.debug(f"Cleaning allocation\_date: input={date\_input}, type={type(date\_input)}")

if not date\_input:

logger.error("allocation\_date is empty")

raise forms.ValidationError(\_('تاریخ تخصیص اجباری است.'))

if isinstance(date\_input, date):

logger.debug(f"allocation\_date is already a date: {date\_input}")

return date\_input

try:

date\_str = to\_english\_digits(str(date\_input).strip())

date\_str = re.sub(r'[-\.]', '/', date\_str)

# بررسی چندین فرمت مختلف برای تاریخ

formats = ['%Y/%m/%d', '%Y-%m-%d', '%Y%m%d']

for fmt in formats:

try:

j\_date = jdatetime.date.strptime(date\_str, fmt)

g\_date = j\_date.togregorian()

if isinstance(g\_date, date):

logger.debug(f"Parsed allocation\_date: {g\_date} with format {fmt}")

return g\_date

except ValueError:

continue

logger.error(f"Invalid allocation\_date format: {date\_input}")

raise forms.ValidationError(\_('لطفاً تاریخ معتبری وارد کنید (مثل 1404/01/17 یا 1404-01-17).'))

except Exception as e:

logger.error(f"Error parsing date: {str(e)}")

raise forms.ValidationError(\_('خطا در پردازش تاریخ. لطفاً فرمت صحیح وارد کنید.'))

def clean(self):

cleaned\_data = super().clean()

logger.debug(f"Running clean method: cleaned\_data={cleaned\_data}")

organization = cleaned\_data.get('organization')

project = cleaned\_data.get('project')

allocated\_amount = cleaned\_data.get('allocated\_amount')

allocation\_date = cleaned\_data.get('allocation\_date')

budget\_period = cleaned\_data.get('budget\_period')

allocation\_type = cleaned\_data.get('allocation\_type')

locked\_percentage = cleaned\_data.get('locked\_percentage')

warning\_threshold = cleaned\_data.get('warning\_threshold')

# اعتبارسنجی دوره بودجه

if not budget\_period:

logger.error("No budget\_period selected")

raise forms.ValidationError(\_("دوره بودجه باید انتخاب شود."))

# اعتبارسنجی سازمان

if not organization:

logger.error("No organization selected")

raise forms.ValidationError(\_('لطفاً یک سازمان انتخاب کنید.'))

# اعتبارسنجی پروژه

if not project:

logger.error("No project selected")

raise forms.ValidationError(\_('لطفاً یک پروژه انتخاب کنید.'))

# اعتبارسنجی مبلغ تخصیص

if allocated\_amount is None or allocated\_amount <= 0:

logger.error(f"Invalid allocated\_amount: {allocated\_amount}")

raise forms.ValidationError(\_('مبلغ تخصیص باید مثبت باشد.'))

# اعتبارسنجی ارتباط سازمان و پروژه

if organization and project:

if not project.organizations.filter(id=organization.id).exists():

logger.error(f"Project {project} does not belong to organization {organization}")

raise forms.ValidationError(\_('پروژه انتخاب‌شده متعلق به این سازمان نیست.'))

# محاسبه بودجه باقیمانده

if budget\_period and allocated\_amount:

used\_budget = BudgetAllocation.objects.filter(

budget\_period=budget\_period

).exclude(id=self.instance.id).aggregate(

total=Sum('allocated\_amount')

)['total'] or Decimal('0')

remaining\_budget = budget\_period.total\_amount - used\_budget

# اگر نوع تخصیص درصد باشد، محاسبه مبلغ معادل

if allocation\_type == 'percent':

allocated\_amount = (allocated\_amount / 100) \* budget\_period.total\_amount

if allocated\_amount > remaining\_budget:

logger.error(f"allocated\_amount ({allocated\_amount}) exceeds remaining\_budget ({remaining\_budget})")

raise forms.ValidationError(\_(

f'مبلغ تخصیص ({allocated\_amount:,} ریال) بیشتر از بودجه باقی‌مانده ({remaining\_budget:,} ریال) است.'

))

# اعتبارسنجی تاریخ تخصیص

if allocation\_date and budget\_period:

start\_date = budget\_period.start\_date

end\_date = budget\_period.end\_date

if allocation\_date < start\_date or allocation\_date > end\_date:

logger.error(f"allocation\_date ({allocation\_date}) outside budget\_period range")

raise forms.ValidationError(\_('تاریخ تخصیص باید در بازه دوره بودجه باشد.'))

# اعتبارسنجی درصدها

if locked\_percentage is not None and (locked\_percentage < 0 or locked\_percentage > 100):

raise forms.ValidationError(\_('درصد قفل‌شده باید بین ۰ تا ۱۰۰ باشد.'))

if warning\_threshold is not None and (warning\_threshold < 0 or warning\_threshold > 100):

raise forms.ValidationError(\_('آستانه اخطار باید بین ۰ تا ۱۰۰ باشد.'))

if locked\_percentage is not None and warning\_threshold is not None:

if locked\_percentage >= warning\_threshold:

raise forms.ValidationError(\_('آستانه اخطار باید بزرگتر از درصد قفل‌شده باشد.'))

logger.debug("Clean method completed")

return cleaned\_data

def save(self, commit=True):

logger.debug("Saving BudgetAllocationForm")

instance = super().save(commit=False)

instance.budget\_period = self.budget\_period

if self.user and self.user.is\_authenticated:

instance.created\_by = self.user

logger.debug(f"Set created\_by: {self.user}")

else:

logger.error("No authenticated user provided for created\_by")

raise forms.ValidationError(\_('کاربر معتبر برای ایجاد تخصیص بودجه لازم است.'))

if commit:

try:

instance.save()

logger.info(f"BudgetAllocation saved: {instance}")

except Exception as e:

logger.error(f"Error saving BudgetAllocation: {str(e)}")

raise

return instance

و فایل توابع

import logging

import jdatetime

from decimal import Decimal

from django.db.models import Sum, Q

from django.core.cache import cache

from Tanbakhsystem.utils import parse\_jalali\_date

from django.contrib.contenttypes.models import ContentType

from django.utils import timezone

from Tanbakhsystem.utils import parse\_jalali\_date

from django.utils.translation import gettext\_lazy as \_

logger = logging.getLogger(\_\_name\_\_)

def calculate\_total\_allocated(entity=None, filters=None):

from budgets.models import BudgetAllocation

if filters is None:

filters = {}

queryset = BudgetAllocation.objects.all()

if entity:

from core.models import Organization

if isinstance(entity, Organization):

queryset = queryset.filter(organization=entity)

else:

logger.warning(f"Invalid entity type for calculate\_total\_allocated: {type(entity)}")

return Decimal('0')

if 'budget\_period' in filters:

queryset = queryset.filter(budget\_period=filters['budget\_period'])

if 'date\_from' in filters:

date\_from = parse\_jalali\_date(filters['date\_from']) if isinstance(filters['date\_from'], str) else filters[

'date\_from']

queryset = queryset.filter(allocation\_date\_\_gte=date\_from)

if 'date\_to' in filters:

date\_to = parse\_jalali\_date(filters['date\_to']) if isinstance(filters['date\_to'], str) else filters['date\_to']

queryset = queryset.filter(allocation\_date\_\_lte=date\_to)

if 'is\_active' in filters:

queryset = queryset.filter(is\_active=filters['is\_active'])

total = queryset.aggregate(total=Sum('allocated\_amount'))['total'] or Decimal('0')

logger.debug(f"calculate\_total\_allocated: entity={entity}, filters={filters}, total={total}")

return total

def calculate\_remaining\_budget(obj):

from budgets.models import BudgetPeriod, BudgetAllocation, ProjectBudgetAllocation

from tankhah.models import Tankhah

cache\_key = f"remaining\_budget\_{type(obj).\_\_name\_\_}\_{obj.pk}"

cached\_result = cache.get(cache\_key)

if cached\_result is not None:

logger.debug(f"Returning cached remaining\_budget for {cache\_key}: {cached\_result}")

return cached\_result

if isinstance(obj, BudgetPeriod):

allocated = BudgetAllocation.objects.filter(budget\_period=obj).aggregate(

total=Sum('allocated\_amount')

)['total'] or Decimal('0')

result = max(obj.total\_amount - allocated, Decimal('0'))

elif isinstance(obj, BudgetAllocation):

used = ProjectBudgetAllocation.objects.filter(budget\_allocation=obj).aggregate(

total=Sum('allocated\_amount')

)['total'] or Decimal('0')

result = max(obj.allocated\_amount - used, Decimal('0'))

elif isinstance(obj, ProjectBudgetAllocation):

consumed = Tankhah.objects.filter(project\_budget\_allocation=obj).aggregate(

total=Sum('amount')

)['total'] or Decimal('0')

result = max(obj.allocated\_amount - consumed, Decimal('0'))

else:

logger.warning(f"Invalid object type for calculate\_remaining\_budget: {type(obj)}")

result = Decimal('0')

cache.set(cache\_key, result, timeout=300)

logger.debug(f"calculate\_remaining\_budget: obj={obj}, result={result}")

return result

def check\_budget\_status(obj):

"""چک کردن وضعیت بودجه"""

from budgets.models import BudgetPeriod

if not isinstance(obj, BudgetPeriod):

logger.warning(f"Invalid object type for check\_budget\_status: {type(obj)}")

return 'unknown', 'وضعیت نامشخص'

cache\_key = f"budget\_status\_{obj.pk}"

cached\_result = cache.get(cache\_key)

if cached\_result is not None:

logger.debug(f"Returning cached budget\_status for {cache\_key}: {cached\_result}")

return cached\_result

remaining = calculate\_remaining\_budget(obj)

locked = obj.get\_locked\_amount()

warning = obj.get\_warning\_amount()

if not obj.is\_active:

result = 'inactive', \_('دوره غیرفعال است.')

elif obj.is\_completed:

result = 'completed', \_('بودجه تمام‌شده است.')

elif remaining <= 0 and obj.lock\_condition == 'ZERO\_REMAINING':

obj.is\_completed = True

obj.is\_active = False

obj.save()

result = 'completed', \_('بودجه به صفر رسیده و تمام‌شده است.')

elif obj.lock\_condition == 'AFTER\_DATE' and obj.end\_date < timezone.now().date():

obj.is\_active = False

obj.save()

result = 'locked', \_('دوره به دلیل پایان تاریخ قفل شده است.')

elif obj.lock\_condition == 'MANUAL' and remaining <= locked:

result = 'locked', \_('بودجه به حد قفل‌شده رسیده است.')

elif remaining <= warning:

result = 'warning', \_('بودجه به آستانه هشدار رسیده است.')

else:

result = 'normal', \_('وضعیت عادی')

# if result in ('completed', 'locked', 'inactive'):

# BudgetHistory.objects.create(

# content\_type=ContentType.objects.get\_for\_model(BudgetPeriod),

# object\_id=obj.pk,

# action='STATUS\_CHANGE',

# details=f"وضعیت به {result} تغییر کرد: {message}"

# )

# result+= status, message

cache.set(cache\_key, result, timeout=300)

logger.debug(f"check\_budget\_status: obj={obj}, result={result}")

return result

def apply\_filters(queryset, filters):

if not filters:

return queryset

if 'date\_from' in filters:

date\_from = parse\_jalali\_date(filters['date\_from']) if isinstance(filters['date\_from'], str) else filters[

'date\_from']

queryset = queryset.filter(allocation\_date\_\_gte=date\_from)

if 'date\_to' in filters:

date\_to = parse\_jalali\_date(filters['date\_to']) if isinstance(filters['date\_to'], str) else filters['date\_to']

queryset = queryset.filter(allocation\_date\_\_lte=date\_to)

if 'is\_active' in filters:

queryset = queryset.filter(budget\_period\_\_is\_active=filters['is\_active'])

if 'budget\_period' in filters:

queryset = queryset.filter(budget\_period=filters['budget\_period'])

logger.debug(f"apply\_filters: filters={filters}, queryset count={queryset.count()}")

return queryset

def get\_budget\_details(entity=None, filters=None):

from budgets.models import BudgetPeriod, BudgetAllocation, ProjectBudgetAllocation

from core.models import Organization, Project, SubProject

cache\_key = f"budget\_details\_{type(entity).\_\_name\_\_}\_{entity.pk if entity else 'global'}"

cached\_result = cache.get(cache\_key)

if cached\_result is not None:

logger.debug(f"Returning cached budget\_details for {cache\_key}: {cached\_result}")

return cached\_result

if isinstance(entity, BudgetPeriod):

total\_budget = entity.total\_amount

total\_allocated = BudgetAllocation.objects.filter(budget\_period=entity).aggregate(

total=Sum('allocated\_amount')

)['total'] or Decimal('0')

remaining = calculate\_remaining\_budget(entity)

status = check\_budget\_status(entity)

elif isinstance(entity, Organization):

total\_budget = BudgetAllocation.objects.filter(organization=entity).aggregate(

total=Sum('allocated\_amount')

)['total'] or Decimal('0')

total\_allocated = calculate\_total\_allocated(entity=entity, filters=filters)

remaining = calculate\_remaining\_budget(entity)

status = get\_budget\_status(entity, filters)

elif isinstance(entity, Project):

total\_budget = ProjectBudgetAllocation.objects.filter(project=entity).aggregate(

total=Sum('allocated\_amount')

)['total'] or Decimal('0')

total\_allocated = calculate\_total\_allocated(entity=entity, filters=filters)

remaining = calculate\_remaining\_budget(entity)

status = get\_budget\_status(entity, filters)

elif isinstance(entity, SubProject):

total\_budget = ProjectBudgetAllocation.objects.filter(subproject=entity).aggregate(

total=Sum('allocated\_amount')

)['total'] or Decimal('0')

total\_allocated = calculate\_total\_allocated(entity=entity, filters=filters)

remaining = calculate\_remaining\_budget(entity)

status = get\_budget\_status(entity, filters)

else:

total\_budget = BudgetPeriod.objects.aggregate(

total=Sum('total\_amount')

)['total'] or Decimal('0')

total\_allocated = calculate\_total\_allocated(filters=filters)

remaining = calculate\_remaining\_budget(filters=filters)

status = get\_budget\_status(None, filters)

details = {

'total\_budget': total\_budget,

'total\_allocated': total\_allocated,

'remaining\_budget': remaining,

'status': status['status'],

'status\_message': status['message']

}

cache.set(cache\_key, details, timeout=300)

logger.info(f"get\_budget\_details: entity={entity}, filters={filters}, details={details}")

return details

def calculate\_allocation\_percentages(allocations):

total\_percentage = Decimal("0")

for allocation in allocations:

allocation.percentage = (

(allocation.allocated\_amount / allocation.budget\_period.total\_amount \* Decimal("100"))

if allocation.budget\_period.total\_amount else Decimal("0")

)

total\_percentage += allocation.percentage

logger.debug(f"calculate\_allocation\_percentages: total\_percentage={total\_percentage}, count={len(allocations)}")

return total\_percentage

def get\_organization\_budget(organization):

total = calculate\_total\_allocated(entity=organization)

logger.debug(f"get\_organization\_budget: org={organization}, total={total}")

return total

def get\_project\_total\_budget(project):

from budgets.models import ProjectBudgetAllocation

total = ProjectBudgetAllocation.objects.filter(project=project, subproject\_\_isnull=True).aggregate(

total=Sum('allocated\_amount')

)['total'] or Decimal("0")

logger.debug(f"get\_project\_total\_budget: project={project}, total={total}")

return total

def get\_project\_used\_budget(project):

from budgets.models import ProjectBudgetAllocation

subproject\_budget = ProjectBudgetAllocation.objects.filter(project=project, subproject\_\_isnull=False).aggregate(

total=Sum('allocated\_amount')

)['total'] or Decimal("0")

tankhah\_budget = project.tankhah\_set.filter(status='PAID').aggregate(

total=Sum('amount')

)['total'] or Decimal("0")

total = subproject\_budget + tankhah\_budget

logger.debug(

f"get\_project\_used\_budget: project={project}, subproject={subproject\_budget}, tankhah={tankhah\_budget}, total={total}")

return total

def get\_project\_remaining\_budget(project):

remaining = get\_project\_total\_budget(project) - get\_project\_used\_budget(project)

logger.debug(f"get\_project\_remaining\_budget: project={project}, remaining={remaining}")

return remaining

def get\_subproject\_total\_budget(subproject):

from budgets.models import ProjectBudgetAllocation

total = ProjectBudgetAllocation.objects.filter(subproject=subproject).aggregate(

total=Sum('allocated\_amount')

)['total'] or Decimal("0")

logger.debug(f"get\_subproject\_total\_budget: subproject={subproject}, total={total}")

return total

def get\_subproject\_used\_budget(subproject):

total = subproject.tankhah\_set.filter(status='PAID').aggregate(

total=Sum('amount')

)['total'] or Decimal("0")

logger.debug(f"get\_subproject\_used\_budget: subproject={subproject}, total={total}")

return total

def get\_subproject\_remaining\_budget(subproject):

remaining = get\_subproject\_total\_budget(subproject) - get\_subproject\_used\_budget(subproject)

logger.debug(f"get\_subproject\_remaining\_budget: subproject={subproject}, remaining={remaining}")

return remaining

def can\_delete\_budget(entity):

from core.models import Project, SubProject

if isinstance(entity, Project):

can\_delete = not entity.tankhah\_set.exists() and not entity.subprojects.exists()

elif isinstance(entity, SubProject):

can\_delete = not entity.tankhah\_set.exists()

else:

can\_delete = False

logger.debug(f"can\_delete\_budget: entity={entity}, can\_delete={can\_delete}")

return can\_delete

def get\_locked\_amount(obj):

from budgets.models import BudgetPeriod

if isinstance(obj, BudgetPeriod):

return (obj.total\_amount \* obj.locked\_percentage) / Decimal('100')

return Decimal('0')

def get\_warning\_amount(obj):

from budgets.models import BudgetPeriod

if isinstance(obj, BudgetPeriod):

return (obj.total\_amount \* obj.warning\_threshold) / Decimal('100')

return Decimal('0')

def get\_budget\_status(entity, filters=None):

"""

تابع وضعیت بودجه برای سازمان‌ها، پروژه‌ها، و زیرپروژه‌ها

فرض: مشابه check\_budget\_status اما برای موجودیت‌های دیگر

"""

"""بررسی وضعیت بودجه"""

from budgets.models import BudgetPeriod, BudgetAllocation, ProjectBudgetAllocation

from core.models import Organization, Project, SubProject

cache\_key = f"budget\_status\_{type(entity).\_\_name\_\_}\_{entity.pk}"

cached\_result = cache.get(cache\_key)

if cached\_result is not None:

logger.debug(f"Returning cached budget\_status for {cache\_key}: {cached\_result}")

return cached\_result

if isinstance(entity, BudgetPeriod):

status, message = entity.check\_budget\_status()

elif isinstance(entity, Organization):

allocations = BudgetAllocation.objects.filter(organization=entity)

if filters:

allocations = apply\_filters(allocations, filters)

if not allocations.exists():

status, message = 'no\_budget', 'هیچ بودجه‌ای تخصیص نیافته است.'

else:

active\_count = allocations.filter(budget\_period\_\_is\_active=True).count()

status = 'active' if active\_count > 0 else 'inactive'

message = f"{active\_count} تخصیص فعال از {allocations.count()} کل"

elif isinstance(entity, (Project, SubProject)):

allocations = ProjectBudgetAllocation.objects.filter(

Q(project=entity) if isinstance(entity, Project) else Q(subproject=entity)

)

if not allocations.exists():

status, message = 'no\_budget', 'هیچ بودجه‌ای تخصیص نیافته است.'

else:

active\_count = allocations.filter(budget\_allocation\_\_budget\_period\_\_is\_active=True).count()

status = 'active' if active\_count > 0 else 'inactive'

message = f"{active\_count} تخصیص فعال از {allocations.count()} کل"

else:

status, message = 'unknown', 'وضعیت نامشخص'

result = {'status': status, 'message': message}

cache.set(cache\_key, result, timeout=300)

logger.debug(f"get\_budget\_status: entity={entity}, result={result}")

return result

حالا تغییرا قبلی اعمال شده

ببین سناریو درسته ؟

عالی! با داشتن همه مدل‌ها، می‌توانیم تصویر کامل‌تری از سیستم و نحوه تعامل مدل‌ها برای ایجاد گردش کار تایید/رد داشته باشیم.

بیایید مدل‌ها و ارتباطاتشان را با تمرکز بر فرآیند تایید/رد بررسی کنیم:

**1. بازیگران و ساختار سازمانی:**

* **CustomUser**: کاربر نهایی سیستم.
* **Organization, OrganizationType**: ساختار کلی سازمان (دفتر مرکزی، شعبات).
* **Post**: نقش‌های سازمانی با سلسله مراتب (parent, level), شاخه (branch) و سازمان (organization) مشخص. فیلد is\_payment\_order\_signer یک مجوز خاص برای امضای دستور پرداخت است.
* **UserPost**: اتصال کاربر به پست سازمانی. این اتصال تعیین می‌کند کاربر چه نقشی دارد و در نتیجه چه اختیاراتی در گردش کار خواهد داشت.
  + **نکات مثبت:** ساختار سلسله مراتبی پست‌ها و اتصال کاربر به پست واضح است.
  + **نیاز به توجه:** هنگام بررسی اختیارات کاربر، باید UserPost فعال (با is\_active=True و end\_date=None) در نظر گرفته شود.

**2. تعریف گردش کار (Workflow Definition):**

* **WorkflowStage**: مراحل فرآیند را با order مشخص می‌کند. is\_final\_stage برای تشخیص پایان فرآیند تایید اصلی بسیار مهم است.
* **StageApprover**: به وضوح مشخص می‌کند *کدام پست سازمانی* (post) در *کدام مرحله* (stage) می‌تواند اقدام کند. unique\_together جلوی تعریف تکراری را می‌گیرد.
* **PostAction**: (این مدل بسیار کلیدی است) تعریف می‌کند که یک Post در یک Stage خاص، *چه نوع اقدامات مشخصی* (action\_type) می‌تواند انجام دهد (مثل تایید، رد، صدور دستور پرداخت). این به شما کنترل دقیق می‌دهد.
* **TankhActionType**: به نظر می‌رسد لیستی از انواع اقدامات ممکن را تعریف می‌کند (شاید جایگزین یا تکمیل کننده PostAction.ACTION\_TYPES باشد).
* **TransactionType**: برای دسته‌بندی هزینه‌ها (مثل بیمه، جریمه) و تعیین نیاز به تایید اضافی (requires\_extra\_approval) یا حداقل مرحله مجاز (FactorItem.min\_stage\_order).
  + **نکات مثبت:** پایه‌های تعریف گردش کار (مراحل، مسئول هر مرحله، اقدامات مجاز در هر مرحله) وجود دارد.
  + **نیاز به توجه/ابهام:**
    - **همپوشانی PostAction و TankhActionType/TankhahAction**: نقش دقیق TankhActionType و TankhahAction در کنار PostAction چیست؟ آیا PostAction قوانین کلی را تعریف می‌کند و TankhahAction اقدامات خاص روی تنخواه را ثبت می‌کند؟ این قسمت نیاز به شفاف‌سازی دارد تا از افزونگی جلوگیری شود. به نظر می‌رسد PostAction برای تعریف *قوانین* و TankhahAction یا ApprovalLog برای ثبت *وقایع* استفاده می‌شوند که منطقی است.
    - **استفاده از TransactionType در FactorItem**: اتصال FactorItem.transaction\_type به TransactionType و داشتن min\_stage\_order در FactorItem برای کنترل اینکه چه نوع هزینه‌ای در چه مرحله‌ای قابل ثبت است، بسیار خوب است.

**3. آیتم‌های در گردش (Items Being Processed):**

* **BudgetPeriod, BudgetAllocation, BudgetItem**: برای مدیریت بودجه کلان، تخصیص به سازمان‌ها/پروژه‌ها و ردیف‌های بودجه. خود اینها ممکن است نیاز به تایید داشته باشند (مثلاً ایجاد BudgetPeriod یا BudgetAllocation).
* **Project, SubProject, ProjectBudgetAllocation**: پروژه‌ها و تخصیص بودجه به آنها.
* **Tankhah**: درخواست اصلی تنخواه که در گردش کار حرکت می‌کند. دارای status و current\_stage است.
* **Factor, FactorItem**: جزئیات هزینه‌های تنخواه. FactorItem دارای status مجزا است که امکان تایید/رد در سطح ردیف را می‌دهد.
* **PaymentOrder**: خروجی نهایی فرآیند تایید (معمولاً). دارای status و min\_signatures است.
  + **نکات مثبت:** مدل‌ها به خوبی آیتم‌های مالی مختلف را پوشش می‌دهند و فیلدهای لازم برای پیگیری وضعیت (status) و مرحله (current\_stage) را دارند. اتصال بودجه به تنخواه (Tankhah.budget\_allocation) و مصرف آن از طریق BudgetTransaction مهم است.
  + **نیاز به توجه:**
    - **فیلدهای approved\_by**: فیلدهای approved\_by در Tankhah و Factor (که ManyToManyField به CustomUser هستند) با وجود ApprovalLog به نظر **اضافی (redundant)** می‌آیند. ApprovalLog اطلاعات کامل‌تری (چه کسی، کی، در چه مرحله‌ای، با چه پستی، چه اقدامی) ثبت می‌کند. توصیه می‌شود این فیلدها حذف شوند تا از ناهماهنگی جلوگیری شود.
    - **منطق به‌روزرسانی status و current\_stage**: این منطق در خود مدل‌ها وجود ندارد و باید در لایه دیگری (View یا Service) پیاده‌سازی شود که بر اساس اقدامات ثبت شده در ApprovalLog این فیلدها را تغییر دهد.
    - **قفل شدن (is\_locked, locked)**: مدل‌های Tankhah و Factor فیلدهای قفل دارند. منطق اینکه چه زمانی و توسط چه کسی قفل می‌شوند باید پیاده‌سازی شود (مثلاً در مرحله نهایی یا توسط مدیر).

**4. ثبت تاریخچه و اقدامات (Logging & History):**

* **ApprovalLog**: برای ثبت اقدامات تایید/رد/تغییر مرحله روی Tankhah, Factor, FactorItem. این مدل بسیار مهم است و به عنوان تاریخچه اصلی گردش کار عمل می‌کند. فیلدهای user, post, stage, action, timestamp, comment را دارد.
* **TankhahAction**: به نظر می‌رسد برای ثبت اقدامات خاص‌تر روی تنخواه (شاید فراتر از تایید/رد ساده، مثل "صدور دستور پرداخت" یا "ثبت بیمه") باشد. همانطور که گفته شد، ارتباطش با ApprovalLog و PostAction باید شفاف شود.
* **BudgetHistory**: برای ثبت تاریخچه تغییرات در بودجه و تخصیص‌ها. از ContentType استفاده می‌کند اما GenericForeignKey ندارد (که اشکالی ندارد اگر فقط برای مدل‌های بودجه استفاده شود).
* **PostHistory**: برای ثبت تغییرات در پست‌های سازمانی.
  + **نکات مثبت:** مدل‌های خوبی برای ثبت تاریخچه وجود دارد. ApprovalLog به درستی به WorkflowStage, CustomUser, Post و آیتم‌های مورد تایید لینک شده است. ApprovalLog.save شامل بررسی دسترسی بر اساس PostAction است که عالی است.
  + **نیاز به توجه:**
    - **پر کردن تاریخچه‌ها**: منطق ایجاد رکوردهای BudgetHistory و PostHistory باید در زمان انجام تغییرات مربوطه (مثلاً در save مدل‌های BudgetAllocation یا Post یا با استفاده از Signals) پیاده‌سازی شود. BudgetHistory در حال حاضر فقط برای RETURN در BudgetTransaction پر می‌شود.
    - **شفاف‌سازی ApprovalLog در مقابل TankhahAction**: آیا هر دو لازم هستند؟ اگر TankhahAction برای اقدامات خاص است، شاید ApprovalLog فقط برای تایید/رد/تغییر مرحله استفاده شود.

**5. فرآیند پرداخت:**

* **PaymentOrder**: مدل دستور پرداخت که می‌تواند از Tankhah و Factor ها ایجاد شود.
* **Payee**: اطلاعات دریافت‌کننده وجه.
  + **نیاز به توجه:**
    - **ایجاد PaymentOrder**: منطق ایجاد PaymentOrder پس از رسیدن Tankhah به مرحله نهایی تایید باید پیاده‌سازی شود.
    - **امضاها (min\_signatures)**: اگر min\_signatures > 1 باشد، نیاز به مکانیزمی برای ثبت امضاهای متعدد است (مثلاً یک مدل PaymentOrderSignature که به PaymentOrder و ApprovalLog یا User/Post لینک شود).

**جمع‌بندی و توصیه‌ها:**

شما یک مجموعه مدل نسبتاً کامل و خوب برای سیستم مالی و گردش کار طراحی کرده‌اید. ارتباطات اصلی بین مدل‌ها منطقی به نظر می‌رسد.

**مهم‌ترین گام‌های بعدی و بهبودها:**

1. **پیاده‌سازی منطق گردش کار (Workflow Engine Logic):** این مهم‌ترین بخش است که وجود ندارد. شما نیاز به کد (در Views، Services یا یک اپ جداگانه workflow) دارید که:
   * درخواست‌های کاربر (کلیک روی دکمه تایید/رد) را دریافت کند.
   * دسترسی کاربر را بر اساس Post, WorkflowStage, StageApprover, و PostAction بررسی کند.
   * یک رکورد در ApprovalLog (و شاید TankhahAction بسته به تصمیم شما) ایجاد کند.
   * فیلدهای status و current\_stage را در مدل مربوطه (Tankhah, Factor, ...) بر اساس نتیجه اقدام (تایید/رد) و WorkflowStage.order به‌روز کند.
   * اقدامات جانبی را انجام دهد (مثلاً ارسال Notification یا ایجاد PaymentOrder در مرحله نهایی).
2. **شفاف‌سازی و حذف افزونگی:**
   * نقش دقیق PostAction, TankhActionType, TankhahAction و ApprovalLog.action\_type را مشخص کنید و مدل‌ها یا فیلدهای اضافی را حذف یا ادغام کنید. پیشنهاد اولیه: PostAction برای تعریف قوانین، ApprovalLog برای ثبت وقایع تایید/رد/تغییر مرحله، و TankhahAction (اگر لازم است) برای ثبت اقدامات خاص سیستمی مثل "صدور دستور پرداخت".
   * فیلدهای approved\_by در Tankhah و Factor را حذف کنید و به ApprovalLog تکیه کنید.
3. **تکمیل تاریخچه‌ها:**
   * منطق ایجاد PostHistory را (ترجیحاً با Signals یا override کردن Post.save) پیاده‌سازی کنید.
   * منطق ایجاد BudgetHistory را برای اقدامات CREATE, UPDATE, STOP, REALLOCATE تکمیل کنید.
4. **پیاده‌سازی سیستم امضا برای PaymentOrder:** اگر به بیش از یک امضا نیاز دارید، مکانیزم آن را طراحی و پیاده‌سازی کنید.
5. **بررسی محاسبات remaining\_amount**: اطمینان حاصل کنید که نحوه محاسبه و به‌روزرسانی remaining\_amount در BudgetAllocation و ProjectBudgetAllocation با توجه به BudgetTransaction ها کاملاً دقیق و هماهنگ است.

با انجام این مراحل، مدل‌های شما به یک سیستم گردش کار کارآمد و قابل اتکا تبدیل خواهند شد.